TOWARDS METRIC-DRIVEN, APPLICATION-SPECIFIC VISUALIZATION OF ATTACK GRAPHS

MICKAEL EMIRKANIAN-BOUCHARD

A THESIS
IN
THE DEPARTMENT
OF
CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF APPLIED SCIENCE (INFORMATION SYSTEMS SECURITY)
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

AUGUST 2013

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By:	**Mickael Emirkanian-Bouchard**
Entitled:	**Towards Metric-Driven, Application-Specific Visualization of Attack Graphs**

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

Master of Applied Science (Information Systems Security)

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

Signed by the final examining commitee:

Dr. Chun Wang	Chair
Dr. Otomane Ait Mohamed (ECE)	External Examiner
Dr. Mohammad Mannan (CIISE)	CIISE Examiner
Dr. Lingyu Wang	Supervisor

Approved ______________________________
Chair of Department

__________ 20 ______ ______________________________
Robin Drew, Dean
Faculty of Engineering and Computer Science

Abstract

Towards Metric-Driven, Application-Specific Visualization of Attack Graphs

Mickael Emirkanian-Bouchard

As a model of vulnerability information, attack graphs have seen successes in many automated analyses for defending computer networks against potential intrusions. On the other hand, attack graphs have long been criticized for their poor scalability when serving as a visualization model for human analysts to comprehend, since even a small network may yield an overly complex and incomprehensible attack graph. In this thesis, we propose two novel approaches to improving attack graph visualization. First, we employ recent advances in network security metrics to design metric-driven visualization techniques, which render the most critical information (with the highest metric scores) the most highlighted or magnified. Second, we observe that existing techniques usually aim at a one-size-fits-all solution, which actually renders them less effective for specific applications, and hence we propose to design application-specific visualization solutions. In this thesis, we focus on two such solutions, for network overview and situational awareness, respectively. We present the model and algorithms, describe our implementation, and present our simulation results with regards to scalability and performance.

Acknowledgments

First and foremost I would like to thank my supervisor, Dr. Lingyu Wang for giving me the opportunity, guidance and support required for conducting research work, solving interesting problems and learning valuable life lessons along the way. I would also like to thank my lab mates, who have always been kind, helpful and eager to learn. Finally I would like to thank my family and friends, whose support was crucial to the completion of these studies.

Contents

List of Figures **vii**

List of Tables **ix**

1 Introduction **1**

2 Background and Related Work **5**

2.1 Background 5

2.1.1 Attack Graphs and Visualization Scalability Issues 5

2.1.2 Security Metrics 9

2.2 Related Work 14

2.2.1 Security Metrics and Attack Graphs 14

2.2.2 Information Visualization 16

2.2.3 Attack Graph Visualization 20

2.2.4 Cyber-Situational Awareness 22

3 Developing Metrics and Applying Existing Visualization Models **23**

3.1 The Asset Value Security Metric 23

3.1.1 Hierarchical Metrics for Visualization through Aggregation 24

3.1.2 Example Application of the Metric 26

3.2 Applying Existing Visualization Models 30

3.2.1 Clustering Attack Graphs 30

3.2.2 Attack Treemaps 32

3.2.3 Hyperbolic Trees 35
3.2.4 Limitations of Existing Visualizations 37

4 Radial Attack Treemaps **39**
4.1 Overview 39
4.2 Models and Algorithms 40
4.2.1 Data Structures 42
4.2.2 Algorithms 43
4.3 Implementation and Simulation 46
4.3.1 Implementation Details 46
4.3.2 Simulation Setup 49
4.3.3 Simulation Results 50
4.4 Discussions and Limitations 52

5 Topographic Hyperbolic Trees **54**
5.1 Overview 54
5.2 Model and Algorithms 56
5.2.1 Data Structures 56
5.2.2 Algorithms 57
5.3 Implementation and Simulation 58
5.3.1 Implementation Details 59
5.3.2 Simulation Setup 59
5.3.3 Simulation Results 62
5.4 Discussion and Limitations 62

6 Conclusion **65**

Bibliography **66**

List of Figures

1 An Example Network Configuration 6

2 Attack Graph Corresponding to the Example in Figure 1 7

3 Attack Graph Scalability Issues 9

4 Bayesian-Network based Metrics for Attack Graphs 13

5 Example of Visalert [LAM$^+$05] 18

6 Example of Circos [KSB$^+$09] 19

7 Example of Netmap [GS06] 19

8 Example of Hierarchical Edge Bundling [Hol06] 19

9 Example of Hierarchical Aggregation [NJ04] 20

10 Example of Clustered Adjacency Matrices[NJ05] 21

11 Example of GARNET [WLI08a] 21

12 Example of NAVIGATOR[CIL$^+$10] 22

13 Host Sub-Graphs to Compute Conditional Probabilities 27

14 Graph Illustrating the Asset Value Security Metric 28

15 Example of a Balloon Attack Graph Using JUNG[OFWB03] 31

16 Treemap Example Using JIT[Bel] 35

17 A Hyperbolic Tree using JIT [Bel] 37

18 A Mockup of the Radial Attack Treemap 41

19 Radial Attack Treemap representing Figure 19 42

20 Link Generation 47

21 A Larger Example of a Radial Attack Treemaps 48

22 Simulation Software Setup 49

23 Average of Edges and Nodes per Host 50
24 Average Size of Attack Graphs and Radial Attack Treemaps 51
25 Average Number of Edges and Links 52
26 Topographic Hyperbolic Tree 55
27 Example of a Topographic Hyperbolic Tree With Varying Centers 60
28 Simulation Software Setup 61
29 Average Number of Nodes Relative to Tree Depth for Various Network Sizes 63
30 Average Computational Complexity in Function of Network Size and Tree Depth 64

List of Tables

1 The Conditional Probability Tables for Figure 4 12

2 Asset Values for the Example Network 26

3 Bayesian Network Conditional Probability Adjacency Matrix 26

4 Different Tiling Algorithms for Treemaps 33

5 Limitation of Existing Visualizations 38

Chapter 1

Introduction

Computer networks have long become the nerve system of enterprise information systems and critical infrastructures, and as such, are vulnerable to outside attacks. The scale and severity of security threats to computer networks have continued to grow at an ever-increasing pace, with high-profile attacks reportedly targeting industrial control systems and military satellites.

To defend computer networks against potential attacks, an important starting point is to understand their flaws and weaknesses, through the assessment, analysis and maintenance of their security. To that end, a network security administrator or analyst should be capable of assessing the security posture of a network quickly and efficiently at all times, should be aware of the network's weaknesses, and the actions which need to be performed in order to harden the network. Comparing the security ramifications of different potential configurations is also of significant importance. However, the amount of vulnerability information in a network increases very rapidly depending on the network's size, mostly because vulnerabilities are seldom independent and attackers may combine them in sophisticated ways for attack propagation or privilege escalation. Therefore, conveying a large amount of vulnerability information to human analysts through an intuitive and scalable system is a challenging issue in most networks.

Attack graph is an established model of vulnerability information in networks [AWK02, SHJ+02], through the aggregation of host and vulnerability data in a semantically-rich fashion. By encoding potential exploits of vulnerabilities and linking them through their common pre- and post-conditions, an attack graph provides a clear picture about how attackers may potentially break into a network and consequently follow certain attack paths to compromise network assets. Attack graphs have seen successes in many automated analyses for assessing, monitoring, and hardening computer networks. The increasing tend towards cloud computing intoduces new challenges, where the reliance on large scale server farms naturally leads to complex relationships, even within a single machine, making it difficult to segregate different virtualized systems within a host system. In such a scenario, attack graphs generated would be quite large, and separating them would compromise the value of this analysis.

For a human analyst, correctly and efficiently extracting the significance of large amounts of data is crucial. A human's vision is his sensory system that provides the most bandwidth [War12], making information visualization the ideal solution to convey this data. By relying on a user's perception and cognition, information visualization is a tool used by a human to offload some of his memory to the visuals and providing him with a working set of information. These visuals are created through the abstraction and the encoding of data using different principles such as: proximity, similarity, continuity, symmetry, closure, size and color.

However, as attack graphs are direct representations of an in-memory model, they have long been criticized for their poor scalability when serving as a visualization model for human analysts to comprehend. Attack graph nodes and edges allow for their algorithmic traversal, with no regard for human perception or cognition. For larger networks, their unmanageable size, poor scalability and unintuitive layout make them difficult to read and understand, with human analysts being limited by their perception, comprehension as well as errors. Better visualization schemes would drastically improve an analyst's comprehension of attack graphs, thus better understanding the network's security posture and its potential weaknesses. It is only by ensuring the synergy between a computer's automatic analysis and a human's intervention that we can fully leverage the power of the attack graph model and guarantee effective, efficient and long lasting network security solutions.

The visualization of attack graphs has received limited attention (a more detailed review of related work will be given in Chapter 2.2). The focus of attack graph visualization research has mostly been for the purpose of reachability analysis. Most efforts to improve scalability are often a trade-off with detail - abstracting nodes and using interactive processes to obtain details - a general loss of information and context for the user. The hierarchical aggregation framework [NJ04] abstracts and hides lower-level details through interactive processes, creating a loss of information for the user. This however only partially improves the scalability since it still relies on the node-link representation of basic attack graphs, and the expression of vulnerability metrics remains difficult. The so-called clustered adjacency matrices [NJ05] address the scalability issues for displaying host reachability, but the highly abstract model renders it unsuitable for human analysts to comprehend, and a display of vulnerability metrics is difficult. GARNET[WLI08a] and NAVIGATOR[CIL+10] employ tree-based structures to represent host configuration, but the display of connectivity and exploit relationships is limited.

Most visualization efforts have been trying to improve the state of attack graph visualization through a single solution. As attack graphs possess a very wide range of potential applications, it is difficult to find a visualization model that fulfills every single requirement for all possible applications. The key is designing visualizations optimized for specific applications.

In this thesis, we propose two novel approaches to improving attack graph visualization, mainly through the management of attack graph complexity through the use of visualization paradigms. First, we employ recent advances in network security metrics to design *metric-driven visualization* techniques. Such techniques prioritize the visualization based on relative metric scores. We will first present metrics to quantify network security through the hierarchical aggregation of exploits and hosts for structured, scalable visualizations. This will allow the most critical information to be most highlighted or magnified in order to guide human analysts to explore the most pertinent threats. Second, we observe that most existing attack graph visualization techniques aim for a one-size-fits-all solution, which actually renders them less effective for specific applications. In this thesis, we propose to design *application-specific visualization* solutions, by initially focusing on two specific use cases or applications:

- The first application is network overview, where the use of vulnerability metrics allow the quantification of a network's security, as *you can't improve what you cannot measure* [Jaq07]. In this scenario, we believe a metric-driven solution is appropriate, where the entire network can be seen at a glance, and each host and vulnerability displayed in a *radial attack treemap*.

- The second application is real-time intrusion detection and cyber-situational awareness, where we believe that the ability able to track and model the attacker's behaviour will facilitate incident response and attack mitigation, while further improving detection of future attacks. For this particular application, we propose an interactive, metric and event-driven solution: *topographic hyperbolic trees*.

We will present both these models and their algorithms, describe our implementations, and present our preliminary simulation results with regards to the performance and scalability of these proposed methods.

The remainder of the thesis is organized as follows. First, Chapter 2.2 will review relevant literature and related work. For the purpose of making this thesis self-contained, Chapter 2.1 will present background information on attack graph and security metrics. Chapter 3.1 will introduce a hierarchical metric framework to further improve vulnerability metric visualization in attack graphs. Building on the strengths and weaknesses of existing techniques reviewed in Chapter 3.2, we will introduce two novel attack graph visualization models for network overview and situational awareness in Chapters 4 and 5, respectively. Finally, Chapter 6 will conclude the thesis.

Chapter 2

Background and Related Work

2.1 Background

For this thesis to be self-contained, we briefly review some background knowledge regarding attack graphs and security metrics, which will be necessary for further discussions. Though some concepts have been introduced in Chapter 2.2, we will go into further detail in this chapter.

2.1.1 Attack Graphs and Visualization Scalability Issues

The Attack Graph Model

An attack graph models vulnerabilities and their inter-dependencies inside a network [AWK02, SHJ+02]. It can be represented as a directed acyclic graph (DAG) composed of two types of nodes, *exploits* and *conditions*, and two types of vertices. The first type of vertex represents the *require* condition: from a condition to an exploit, expressing a pre-condition required by an attacker to execute the exploit. The second type of vertex represents the *imply* condition: from an exploit to a condition - expressing that a post-conditions becomes valid for the attacker once he has successfully executed the exploit.

What differentiates attack graphs from typical directed acyclic graphs is the nature of the *require* relation R_r and the *imply* relation R_i. The former is *conjunctive* while the latter

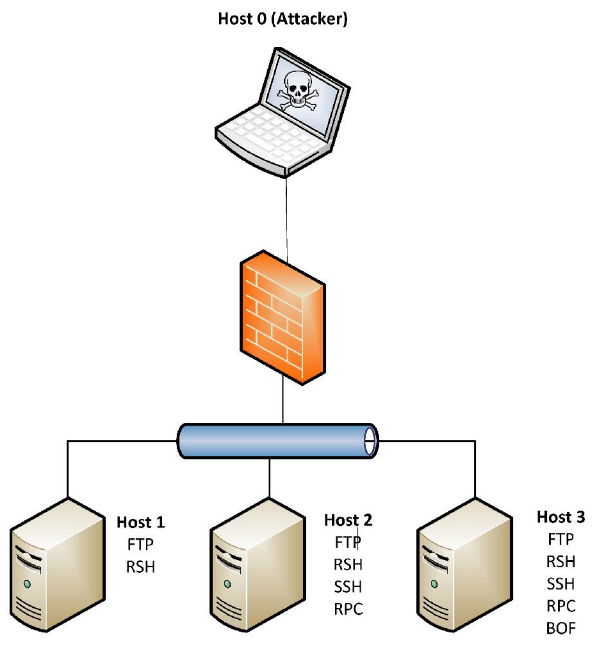


Figure 1: An Example Network Configuration

is *disjunctive*. In other words, for an exploit to be executed, *all* pre-conditions must be satisfied. However, for any post-condition, only *one* of the exploits leading to this condition must be executed for this condition to be satisfied. Definition 1 formally defines an attack graph. While attack graphs can enumerate all possible attack paths, the monotonicity assumption stipulates that an attacker never relinquishes an obtained capability [WNJ06] - every condition gained by an attacker cannot be removed, and therefore does not need to be obtained again.

Figure 1 shows the network configuration of our running example, which will be used throughout the thesis to illustrate different attack visualization methods. The corresponding attack graph shown in Figure 2, in which each predicate *vulnerability (source host, destination host)* inside an oval node indicates a self-explanatory exploit, and each plain-text node *condition(host)* represents a security-related condition. Edges point either from an exploit's pre-conditions (that is, conditions required for executing the exploit) to the exploit, or from the exploit to its post-conditions (that is, conditions implied by executing the exploit). More formally,

Definition 1 (Attack Graph). *An attack graph G is a directed graph* $G(E \cup C, R_r \cup R_i)$

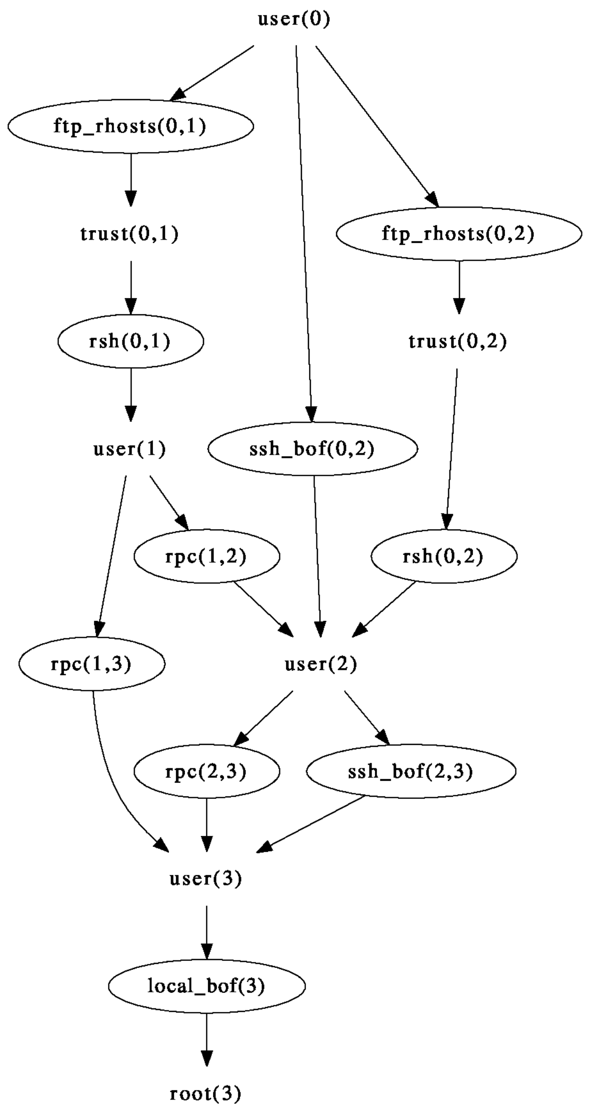


Figure 2: Attack Graph Corresponding to the Example in Figure 1

where E is a set of exploits, C a set of conditions, $R_r \subseteq C \times E$ the require relation and $R_i \subseteq E \times C$ the imply relation.

Attack Graph Visualization and Scalability

The above basic attack graph representation is the straightforward and direct visual interpretation of a software data structure whose purpose is to conduct automated analysis and recursive traversal. It is not particularly well suited for human analysis, as human cognitive processes are very different and significantly more complex than software algorithms.

Since the number of exploits depends on both the amount of vulnerabilities and the pairs of hosts with connectivity, the size of an attack graph will increase very quickly in the size of the network [OBM06]. Enumerating all the exploits, their pre- and post-conditions, and edges between them in a single directed graph will inevitably lead to very high node and edge density, a significant amount of crossings between edges, highly complex edge paths, and a high average edge length. These characteristics render the attack graph messy and difficult to comprehend, and prevent human analysts from interpreting the attack graph and cross validating with results of automated analysis.

Figure 3 shows a messy and illegible attack graph. It has a significant amount of crossings, complex edge paths and a high average edge length. This negatively impacts the readability of these graphs as they appear messy and are difficult to follow and difficult to understand, rendering the analysis of the network difficult and time consuming. It may be surprising to note that this attack graph actually represents a small network composed of only 14 machines, each of which has less than 10 vulnerabilities.

Clearly, the basic representation of attack graphs is not a viable visualization solution. Transferring semantically-rich information from a in-memory computer model to a human analyst through visualization is currently a limiting factor to the utility of the attack graph model.

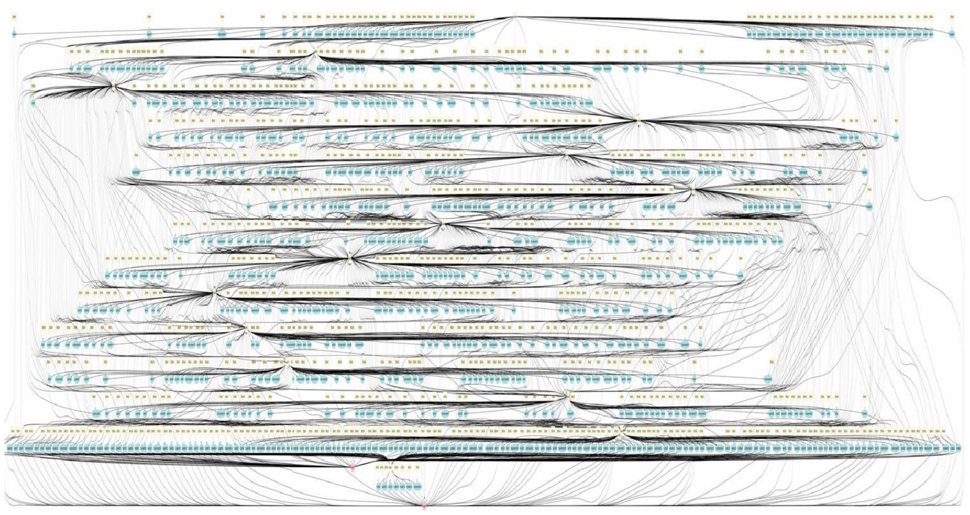


Figure 3: Attack Graph Scalability Issues

2.1.2 Security Metrics

Scoring and ranking vulnerabilities and networks based on their relative severity and security has drawn significant attention lately. Metrics can be applied to attack graphs in order to quantify the relative risk of different network configurations based on known[WIL$^+$08] or unknown[WJSN10] attacks.

The Common Vulnerability Scoring System (CVSS)

The Common Vulnerability Scoring System (CVSS) is a widely recognized standard for security vendors and analysts to assign numerical scores to vulnerabilities to reflect their relative severity [Sch05]. The CVSS score is composed of a *Base Score*, which measures some inherent characteristics of vulnerabilities, such as the exploitability and the impact, a *Temporal Score*, which measures characteristics that may change over time, such as the availability of patches or exploit codes, and an *Environmental Score*, which measures a vulnerability's potential interaction with the surrounding environment. The CVSS scores of vulnerabilities are readily available in public databases, such as the NVD [oST].

The *Base Score* (BS) of a vulnerability assesses factors which are constant over time and environments. It can range from 0 to 10 and is calculated using the following metrics:

- **Access Vector (AV)** : This metric measures what kind access is required to exploit the vulnerability. The possible values can be *Local*, *Adjacent Network* or *Network*.
- **Access Complexity (AC)** : This metric measures the complexity to exploit the vulnerability, which can be either *High*, *Medium* or *Low*
- **Authentication (Au)** : This metric measures the number of times the attacker must authenticate himself to exploit the vulnerability. Possible values are *Multiple*, *Single* and *No*.
- **Confidentiality (C)** : This metric measure a successful exploit's impact on Confidentiality . Possible values are *None*, *Partial*, *Complete*.
- **Integrity (I)** : This metric measures a successful exploit's impact on Integrity . Possible values are *None*, *Partial*, *Complete*.
- **Authentication (A)** : A successful exploit's impact on Availability . Possible values are *None*, *Partial*, *Complete*.

While CVSS is good at measuring metrics for a single exploit, it has certain weaknesses when used in a context where multiple exploits are executed in sequence and share relationships.

Bayesian Network-Based Attack Graphs

A Bayesian network[BG] (or Bayes network or belief network), is a probabilistic statistical graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph in order to represent knowledge about an uncertain domain. For example, a Bayesian network could represent the probabilistic relationships between exploits and conditions. When given conditions, the network can be used to compute the probabilities of the exploitation of various vulnerabilities.

Bayesian networks are represented by graph **G** and parameters **Q**, where **G** is a Directed Acyclic Graph and **Q** is the set of parameters for each node of the network, usually represented by a set Conditional Probability Table (CPT), one for each node. The CPT lists the probability of the different values a child may take depending on its parents. A Bayesian network defines a unique joint distribution represented by:

$$P(X) = \prod_{i=1}^{n} P(x_i | parents(x_i))$$

In other words, X is a Bayesian network if its joint probability density function can be expressed as a product of individual probabilities, conditional on their parents values. Dynamic Bayesian networks are Bayesian networks that represent sequences of variable according to time.
There also exist efforts on combining the CVSS scores of individual vulnerabilities inside a network for an overall measure of the network's security. In particular, the approach in [FW08, FWSJ08] first assigns a normalized CVSS score as the conditional probability of successfully executing each exploit of that vulnerability when their pre-conditions are already satisfied, which reflects the intrinsic difficulty in exploiting a vulnerability. The assigned probabilities are then used to build a Bayesian network based on the causal relationships captured inside the attack graph (e.g., an exploit can only be executed if all of its pre-conditions are satisfied). Finally, the Bayesian network can be used to find the probability that conditions are satisfied and exploits executed inside the attack graph, which provides a security metrics for the whole network.

We apply CVSS-based Bayesian network metrics to the attack graph presented in Figure 2. In the result presented in Figure 4, each number inside an oval represents the conditional probability that the corresponding exploit can be successfully executed when its pre-conditions are already satisfied, whereas each number under a condition indicates the probability of satisfying that condition.

The probability of an attacker executing $ftp_rhosts(0,1)$ is 0.8. Being the only path to the condition $trust(0,1)$, the probability of this condition being satisfied is also 0.8. The probability of the exploit $rsh(0,1)$ is 0.5 and requires the condition $trust(0,1)$ to be satisfied. The probability of $user(1)$, the post-condition of $rsh(0,1)$ is

$$p(trust(0,1)) * p(rsh(0,1)) = 0.8 * 0.5 = 0.4$$

In the case of multiple possible paths towards a given condition, we must calculate the joint probability of reaching this condition through all paths. In the case of the condition $user(2)$ there are three possible ways to satisfy this condition: through the execution of $rpc(1,2)$ with a probability of $0.4 * 0.6 = 0.24$, $ssh\_bof(0,2)$ with a probability of 0.3 or through $rsh(0,2)$ with a probability of 0.4.

$rpc(1,2)$ - A	
T	F
0.24	0.76

$ssh\_bof(0,2)$ - B	
T	F
0.3	0.7

$rsh(0,2)$ - C	
T	F
0.4	0.6

Table 1: The Conditional Probability Tables for Figure 4

We calculate the probability of an attacker reaching the condition $user(2)$ by calculating the probability that one or more exploits leading the condition are executed:

$$\begin{aligned} p(user(2)) &= p(A)[p(B)p(C) + p(B)p(\bar{C}) + p(\bar{B})p(\bar{C}) + p(\bar{B})p(C)] \\ &+ p(\bar{A})[p(B)p(C) + p(B)p(\bar{C}) + p(\bar{B})p(C)] \\ &= 0.68 \end{aligned}$$

Bayesian-network based attack graphs are CVSS-based, and as such, assume that all hosts are equally important ; an attacker taking control of one host is as important as an attacker taking control of any other. As hosts in a network run different services - some which can be more mission-critical than others - and store data with different confidentiality requirements, the same vulnerability exploited on different hosts *should* lead to different metric values, depending on the host's contents and use.

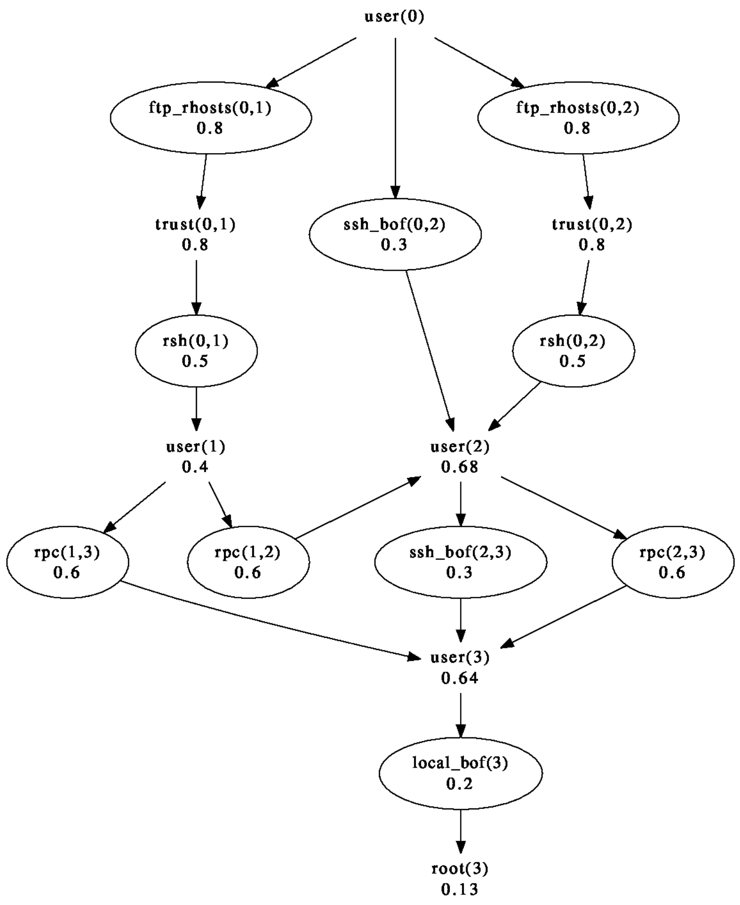


Figure 4: Bayesian-Network based Metrics for Attack Graphs

2.2 Related Work

This chapter is a review of related work in the different fields of study of this paper. The first domain of research is security metrics and attack graphs, the second domain is broadly information visualization and the final deals with attack graph visualization specifically.

2.2.1 Security Metrics and Attack Graphs

Information and systems security is a field in which many complex systems interact, making it difficult to create or identify meaningful metrics.

The National Institute of Standards and Technology (NIST) offers a solid foundation for the understanding and the development of security metrics[Nis03], notably the roles and responsibilities of parties involved in the process, as well as proper procedures to develop and validate new metrics. While this document provides general, high-level information, it provides thorough and valuable background information on the many different aspects of security metrics and metric development.

Jaquith [Jaq07] describes the need for meaningful metrics, and acknowledges the difficulty of estimating asset value and overall risk. Identifying a problem is easy, but assigning a value to the risk is much harder, as there are many different forces at play.

In order to automate security evaluation, researchers use graph structures to model different components pertinent to network security. Schneier[Sch99] introduces attack trees, which model adversary threat behavior for the analysis of more general security systems. The first work on attack graphs in the context of network vulnerability analysis were conducted by Phillips [PS98], where the first work on graph-based methods for risk analysis in computer networks are presented. Since then, the Defense Technical Information Center (DTIC) of the American Department of Defense has sponsored significant amounts of work on attack graphs. Ritchey [RA00] and Ammann [AWK02] first introduce algorithms based on model-checkers by modeling attack scenarios to prove or disprove the security of a given network. By first postulating that the network is secure, the system can verify if the network is indeed secure, or return all counter-examples if the network has been proven to be insecure. Model-checking techniques suffer from scalability issues due to the explosion of possible states, resulting in $O(2^n)$, n the number of nodes (exploits and conditions). The *monotonicity assumption* which states that an attacker never relinquishes any privilege he has gained, has been introduced in [AWK02], reducing the computational complexity to $O(h^2)$.

This work has been extended and formalized further by Sheyner [SHJ$^+$02, SW04, She04] where graph-generating tools have been implemented, and again extended in [JSW02] where the concept of probabilistic attack graphs - interpreting attack graphs as Markov decision processes - has been introduced. Ou et al [OGA05, OBM06] address the previously discussed scalability problems by creating a logical attack graph depicting logical dependencies between network conditions and the attacker's goal. MulVAL[OGA05] (Multi-host, Multi-stage Vulnerability Analysis Language) uses Datalog, a subset of Prolog, to model a computer network in order to perform automatic and efficient ($O(h^2)$) network vulnerability analysis.

Further work has been done to improve attack graphs and further quantify the risk of compromise through different exploits and complex attacker behaviour, as well as the relationship between different exploits in an attack graph. By using a directed acyclic graph of nodes and conditions, Wang et al. [WLJ06, WNJ06] devise a way to leverage attack graphs to employ meaningful metrics, such as CVSS or attacker distance. Attacker distance as a metric was extended for unknown threats in [WJSN10], measuring the resilience of a network to zero-day attacks by counting the number distinct zero-day attacks required to compromise a network asset, further validating the principle of defense-in-depth.

This solution, however, assumes that probabilities along multiple paths leading to a same node are independent, which leads to inaccurate results. Frigault et al. [FW08, FWSJ08] convert acyclic attack graphs into cynamic Bayesian networks but does not permit the handling of cycles. The method presented by Homer et al. [HOS09] handles shared dependencies as well as cycles. Noel et al.[NRJ04] present a method for correlating and mapping IDS data to attack graphs conditions by using attack graph distance to quantify the knowledge and set a threshold for errors.

Idika et al. [IB12] observe that different existing security metrics simply provide a partial view of security, and introduce a suite of combineable attack graph based security metrics which are based on statistics of attack path length. However, a recent study of CVSS-based vulnerability metrics shows the correlation between these metrics and the time to compromise of a system through statistical analysis [HEA].

2.2.2 Information Visualization

Information visualization is the study of the representation of data, and making this representation more suitable for a human's vision and cognition. There is a very a wide range of applications of information visualization across many different scientific fields. Ben Schneiderman's paper [Shn96] is among the most cited works in information visualization, where he presents the *visual information-seeking mantra*, which are guidelines for designing information visualizations: *"Overview first, zoom and filter, then details-on-demand."*. Battista et al. [BETT94] present an annotated bibliography for conventional graph-drawing algorithms. Hermann et al. [HMM00] wrote a survey for Information and Graph Visualization which review the basic visualization paradigms on which most current visualizations today are based upon, most of which will be reviewed in further literature in this chapter.

Visualizing Large Hierarchies

Two main visualizations have been developed to visualize large hierarchies: *treemaps* and *hyperbolic trees*.

Treemaps, introduced by Johnson et al.[JS91], are a graphical representation of a weighted tree by recursively partitioning rectangles depending on the weight assigned to the node. The size of a partition represents the weight of the node, and the color can express another distinct metric. The shape of the partitions is dictated by the *tiling algorithm*, some of which have been reviewed by Shneiderman et al. [SW01]. An example is provided in Chapter 2.1 in Figure 16.

Hyperbolic trees (or hypertrees) are introduced by Lamping et al. [LRP95], a *focus+context* technique - the visualization is centered or focused on a given point, but the context is still visible to the user - which consist in applying a fisheye technique to view and manipulate large hierarchies. This is done by projecting a tree on a hyperbolic plane, using the conformal disk model. An example is provided in Chapter 2.1 in Figure 17.

Radial Visualization

There has recently been a lot of research and interest on radial visualization models in many different scientific fields. Livnat et al. introduced VisAware [LAMF05] is radial visualization system with intent of representing and displaying Situational Awareness in a generalized way. It could be applied. In VisAware, Livnat et al. propose the w^3 *premise* for generalized situational awareness: “What? When? Where”. Three examples are presented, the first is 911 emergency center data, the second is intrusion detection data and the third is alerts regarding biological agents, BioWatch. On the outer ring is displayed the nature of the alert, whether it’s a robbery or an speeding violation in the case of emergency center data, an IDS alert in the case of intrusion detection or a type of biological agent in the case of BioWatch - this is the *what*. The position of this alert on the outer ring represents the time at which the alert has been received - this is the *when*. The center of the ring presents a map - either a network map or a geographical map - which and an edge is drawn between the alert and this point to indicate the *where*. This visualization paradigm is further adapted for intrusion detection systems in VisAlert [LAM$^+$05].

VisAlert [LAM$^+$05] (Figure 5) is a visual intrusion dection correlation system, which seeks to represent the three *w*’s of an IDS alert. The outer ring represents the type of alert - the *what*, the position of this alert on the ring represents the *when* and the inside of the ring is the network’s topology, and an edge going from the type of the alert to a point on this topology represents the *where*. The authors claim it is scalable and enhances situational

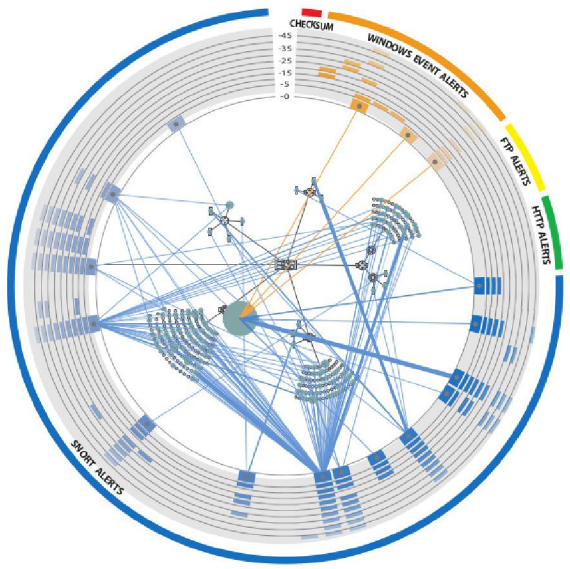


Figure 5: Example of Visalert [LAM+05]

awareness.

Circos [KSB+09](Figure 6) is a radial visualization scheme applied to genomics to facilitate the comparison of genomes as well as displaying similarities. While the data is far from what ag care, we can extract certain similarities: large amounts of data, need to spot outliers and similar elements, scalability is extremely important.

Galloway et al. [GS06] (Figure 7) use a radial visualization in a data mining context. While their dataset was not particularly complex, there was a large amount of edges to visualize, demonstrating the scalability of radial graphs for displaying a very large amount of edges.

As edge scalability is a very strong asset to these radial graphs, further research has been done to improve edges management at the center of these visualizations. Holten et al. [Hol06] introduce a method to hierarchically bundle edges and applies these principles to radial graphs. This is done by creating a tree structure representing the hierarchy in the center and routing edges by these points using Bézier splines. Research by Hong et al. [HN09, HN10] illustrate efforts to find efficient ways to minimize edge crossings in radial graphs.

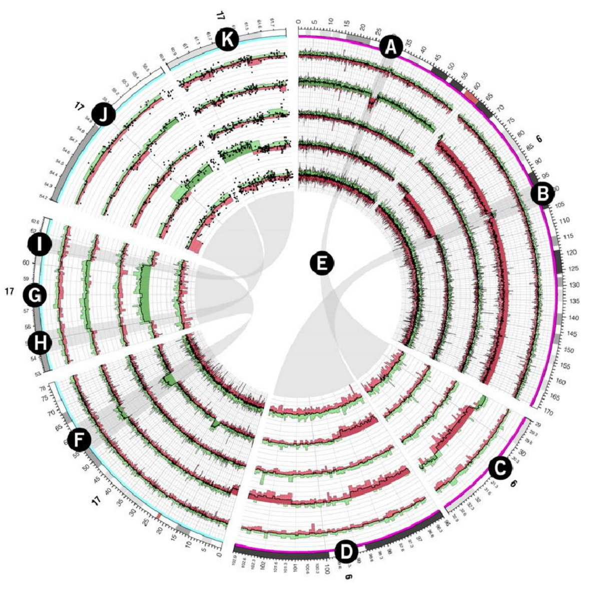


Figure 6: Example of Circos [KSB$^+$09]

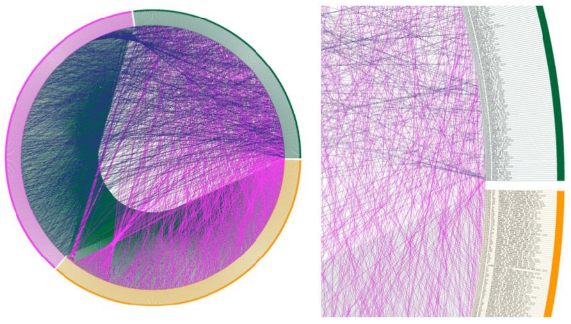


Figure 7: Example of Netmap [GS06]

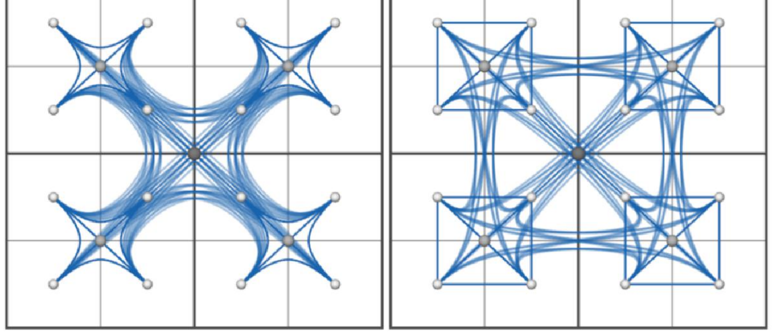


Figure 8: Example of Hierarchical Edge Bundling [Hol06]

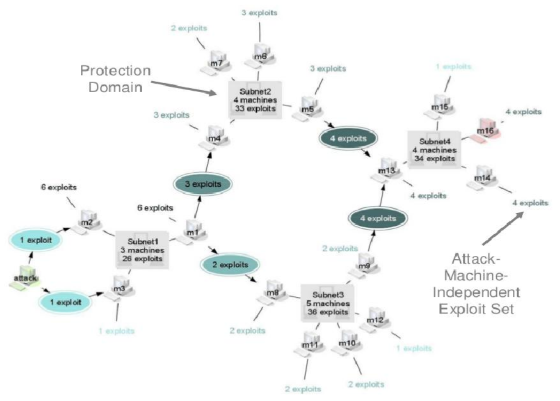


Figure 9: Example of Hierarchical Aggregation [NJ04]

2.2.3 Attack Graph Visualization

Attack graph visualization presents additional challenges due to their specific requirements. Special considerations must be taken to express the logical and physical connections as well as the consequences of the *imply* and *require* condition of conditions and exploits. Here we review literature of visualizationa which are specific to attack graphs.

Most efforts have been made to abstract information from a graph by aggregating nodes or removing and abstracting some attack graph edges in an attempt to "collapse" certain nodes, minimizing density and giving the user the opportunity to obtain more details on demand. Noel et al. [NJ04] present a framework for hierarchically aggregating nodes in an attack graph by using sets of exploits, conditions, *machines* or *protection domains*. Similarly, Homer [HVOM08] et al. developed a method to reduce attack graph complexity by removing "useless" attack steps and collapsing host groups into subnets, via inter and intra-subset trimming of edges.

Noel et al.[NJ05] make use of clustered adjacency matrices to compute the reachability and distance of certain hosts similar to heatmaps[WF09] 10

Some efforts have been made to apply treemaps to attack graphs. GARNET [WLI08a, WLI08b] (Figure 11) is an attack graph visualization tool using the NetSPA (NETwork Security Planning Architecture) [Art02] which displays tremaps [JS91] with semantic substrates [SA06] to visualize a network and it's subnets, as well as reachability of different

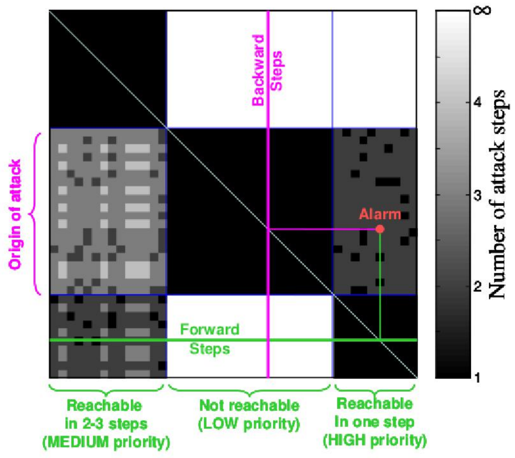


Figure 10: Example of Clustered Adjacency Matrices[NJ05]

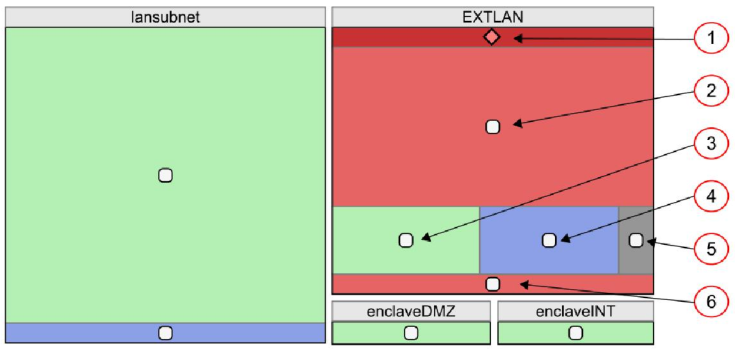


Figure 11: Example of GARNET [WLI08a]

attacks on different subnets or computer groups.

GARNET evolved into NAVIGATOR(Network Asset VIsualization: Graphs, ATtacks, Operational Recommendations) [CIL$^+$10] (Figure 12) , once again using treemaps and semantic substrates. Significant improvements were made, allowing the possibility of zooming-in to the host level and displaying port numbers and vulnerabilities or possible exploits on these ports.

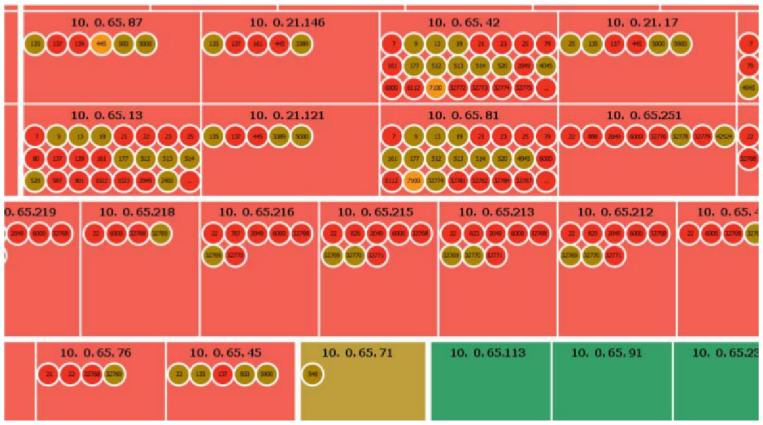


Figure 12: Example of NAVIGATOR[CIL+10]

2.2.4 Cyber-Situational Awareness

Cyber-situational awareness is becoming a strategic area of research in the information security domain, as network attacks share very similar strategies to military operations. However, cyber attacks have unique semantics and their evolution is quicker than in cases of physical attacks[Jaj10]. Jajodia et al.[Jaj10] presents the state of the art in cyber-situational awareness by presenting different technical areas in information security which can contribute to situational awareness, for example machine learning, honeynets, automated vulnerability analysis or topological vulnerability analysis, attack graphs etc

Barford et al.[BDD+10] present the seven main aspects to cyber-situational awareness: situation perception (composed of situation recognition and identification), impact assessment, situation tracking, adversary behavior, causality analysis, data quality (truthfulness, completeness and freshness) and finally plausible futures.

Lakkaraju, Yin et al. present NVisionIP[LYL04] and VisFlowConnect[YYT+04], tools to visualize network traffic in the context of intrusion detection and situational awareness. It does not, however, use the attack graph model and as such, is meant for the user to extract patterns from the traffic, potentially leading to a significant amount false-negatives.

Chapter 3

Developing Metrics and Applying Existing Visualization Models

3.1 The Asset Value Security Metric

Metrics are extremely desirable to network security analysts to measure the severity of potential attacks. William Thomson, also known as Lord Kelvin once said :

> *[...]when you can measure what you are speaking about and express it in numbers you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind[...]* [TK91]

There are vulnerability metrics such as CVSS [Sch05] which evaluate the impact of single vulnerabilities on a given system. Using Bayesian-network based attack graphs allows an analyst to quantify and analyze exploit relationships within a network, but environmental information is not taken into account. Networked systems possess a large number of hosts, and may different services pose a very large array of security requirements. Metrics based purely on CVSS, with no environmental considerations could provide the user with potentially misleading information, as one typically should prioritize network hardening and intrusion defense to critical assets.

Let us suppose that a host is running a mission-critical service, an authentication server for example. The compromise of this host will lead to a substantial loss of availability for services on the network. Another example would be a database server storing sensitive information. The compromise of this asset would lead to a loss of confidentiality through the unauthorized access and possible disclosure of information, but also a potential loss of integrity, if the attacker tampered with the database. Intuitively, a user's workstation would be a much lesser loss, as it is not running mission-critical applications and should not containing highly sensitive data. For this reason, we wish to introduce the notion of *asset value* to attack graphs, by assigning a value between 1 and 10 to each host in the network, depending on it's importance with regards to *confidentiality*, *integrity* and *availability*.

These scenarios demonstrate the need to assign values to hosts, depending on how critical to the network their services are, or on how critical their stored information is to the organization. They also demonstrate the need to quantify the risk one asset may bring to another asset. A host's asset value is user-specified and is based on the following criteria:

1. *Confidentiality Risk*: How serious are the consequences of an attacker obtaining information stored on this host ?

2. *Integrity Risk*: How serious are the consequences of a data loss or data tampering on this host ?

3. *Availability Risk*: How will the loss of this host affect the functioning of the organization's network ?

3.1.1 Hierarchical Metrics for Visualization through Aggregation

In this thesis, we extend the Bayesian network-based security metrics presented by Frigault et al. [FW08, FWSJ08] by introducing the notion of *asset value* to attack graphs, which is a numerical value between 0 and 10 (the domain of CVSS scores) assigned by administrators to each condition in the attack graph based on the condition's relative significance with regards to confidentiality, integrity, and availability. From this assigned asset value, we calculate the *risk* at multiple hierarchical levels for conditions, hosts, group of hosts (subnets), and networks. Here we adopt the common approach of defining risk as the multiplication of the asset value and attack likelihood (that is, the probability obtained using the aforementioned Bayesian network approach). More specifically,

Definition 1. *Given the probability of executing each exploit $p(e)$ and that of satisfying a condition $p(c)$ inside an attack graph $G(E \cup C, R_r \cup R_i)$, and an asset value assignment function $av(.) : C \to [0,10]$, we define Condition Risk, Host Risk and Group Risk as follows :*

Condition Risk

The probability of an exploit being executed $p(e)$ and the probability of a condition being satisfied $p(c)$ are defined in Chapter 2.1.2. The risk of a condition is the probability of this condition being exploited multiplied by the asset value of the host on which this condition applies to.

$$Risk_{Condition}(c) = \frac{p(c) * AssetValue(h)}{10} \tag{1}$$

Host Risk

Host Risk The host risk is the condition risk of the root user-access condition being obtained on this host.

$$Risk_{Host}(h) = Risk_{Condition}(< root, h >) \tag{2}$$

Risk to Another Host The risk of the host h_a to another host h_b is the conditional probability of obtaining access to h_b given that the attacker has compromised h_a.

$$Risk\_To\_Host_b(a) = P(risk_{host}(b)) \mid risk_{host}(a)) \tag{3}$$

Group or Network Risk

For every network or host group, which can either be hosts in close network proximity (eg. subnets) or hosts which share a very similar configuration, host risk is defined as the sum of the host risk of all hosts in this group.

$$Risk_{Group}(G) = \sum_{h \in G} Risk_{Host}(h) \tag{4}$$

3.1.2 Example Application of the Metric

Using the attack graph in Chapter 2.1, Figure 4, we will illustrate the application of the asset value security metric introduced in this chapter, by representing the user-access conditions with their asset values in a directed acyclic graph. We assign the asset values for the host as described in Table 2.

Host	*Value*
$Host_1$	3
$Host_2$	6
$Host_3$	9

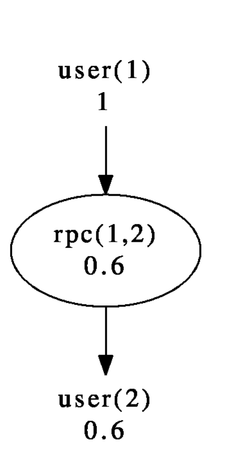
Table 2: Asset Values for the Example Network

For every host condition, we obtain all directly reachable host conditions and calculate the conditional probability of this condition being satisfied. We generate the sub-graphs illustrated in Figure 13 to calculate the conditional probabilities by assigning the probability value of 1 to the host whose risk to others we wish to calculate. The sub-graph from Host 1 to Host 2 is shown in Figure 13a and the one from Host 2 to Host 3 is shown in Figure 13b as examples.

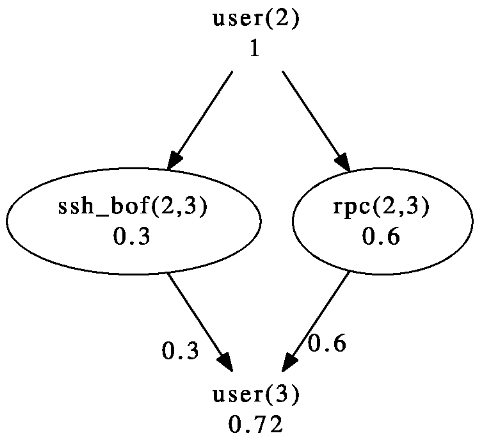
Once the conditional probabilities are calculated, we build the adjacency matrix in which we multiply every column by the asset value of the target host corresponding to this column (Table 2), as illustrated by Table 3.

	$Host_1(3.0)$	$Host_2(6.0)$	$Host_3(9.0)$
$Host_0$	0.12	0.41	
$Host_1$		0.36	0.54
$Host_2$			0.65
$Host_3$			

Table 3: Bayesian Network Conditional Probability Adjacency Matrix



(a) Host 1 to Host 2



(b) Host 2 to Host 3

Figure 13: Host Sub-Graphs to Compute Conditional Probabilities

From this table, we generate a new graph indicating to total risk value of all hosts in the network: the new graph presented in Figure 14 provides us with additional and valuable insight when compared to the initial attack graph. As it is of smaller size, it is easier to rapidly understand the repercussions of network configuration on the security of this network. We can also see the different possible attack paths an attacker can take and with which probability. When compared to the attack graph presented in Chapter 2.1 Figure 4, the score of Host 1 has increased to reflect the potential risk to hosts 2 and 3. Hosts 2 and 3 maintain approximately the same relative score to each other.

To display edge color, we use a color ramping algorithm which returns a color depending on all the scores in the graph. It has already been used in Chapter 3.2 of this thesis for the generation of treemaps. Intuitively, it should start with green, pass by yellow and orange, and finish by red. This algorithm is based on works by Bourke[Bou] and is described in Algorithm 1. $tYellow$ and $tOrange$ are the threshold values for which the color will become yellow or orange respectively. This algorithm will be re-used in Chapters 4 and 5 to further improve attack graph visualization.

The metrics proposed in this chapter serve two purposes :

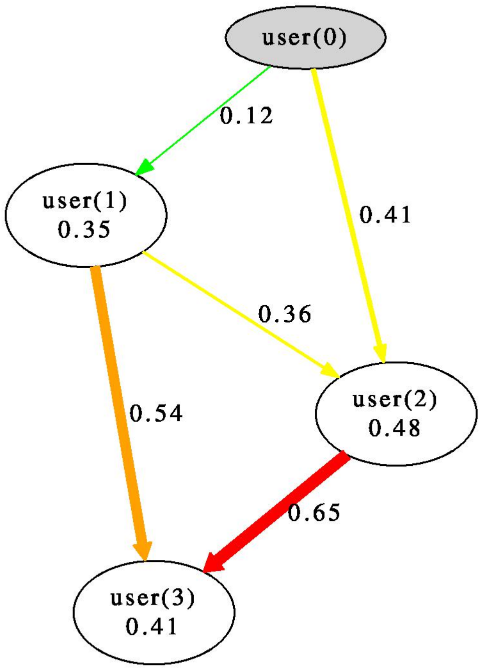


Figure 14: Graph Illustrating the Asset Value Security Metric

Algorithm 1: Color Ramping Algorithm for Normalized Scores Based on [Bou]

```
Input: A score $v$ and a maximal score $vMax$
Output: Returns an 8-bit RGB color c
1  $c.r, c.g, c.b \leftarrow 0$;
2  $normV \leftarrow (v/vMax) * 10$;
3  if $normV < tYellow$ then
4      $c.g \leftarrow 255$;
5  else if $normV < tOrange$ then
6      $c.g \leftarrow 255$;
7      $c.r \leftarrow 255/(tOrange - tYellow) * (normV - tYellow)$;
8  else
9      $c.g \leftarrow 255 - (normV - tOrange) * 255/(10 - tOrange)$;
10     $c.r \leftarrow 255$;
11 return $c$;
```

- To define a metric framework necessary to drive attack graph visualizations. By creating a hierarchical metric framework, it now makes it possible to use hierarchical visualizations and allows for the aggregation of conditions or exploits to hosts, and the aggregation of these hosts to groups of hosts.

- Quantify the risk of an asset being compromised, measuring the probability through a Bayesian network, but also attacker incentive through the use of asset value, but also the risk incurred by this host to others

This allows the attack graph to be built to environmental specifications by a network security administrator, based on which assets are deemed more valuable. These metrics will improve attack graph analysis through effective visualization, and should thus improve network security analyst.

3.2 Applying Existing Visualization Models

In this chapter, we apply and analyze certain existing visualization models to attack graphs using metrics, and analyze their effectiveness, readability and scalability as well as demonstrate their limitations to motivate further discussions. We will continue to use the running example from Chapter 2.1, with the network configuration detailed in Figure 1 and its corresponding attack graph illustrated in Figure 2.

3.2.1 Clustering Attack Graphs

Due to the hierarchical nature of most networks, an obvious approach to improve the scalability of attack graphs is a grouping or clustering certain nodes inside an attack graph that share similar characteristics, such as the same or adjacent hosts [NJ04]. However, such an approach will meet difficulty to maintain readability without losing valuable information due to the relatively high edge density and crossings in a usually highly connected attack graph.

Ballon-Clustered Attack Graphs

The classic top-down graph drawing approach does not facilitate clustering of related information which is not explicitly linked in the data structure. In some situations, the hierarchy of can negatively affect the layout. Melancon et al. [MH98] describe a layout which allows corresponding items in a tree to be grouped together, despite the hierarchical structure requirements. We will now define the *balloon layout* [MH98, LY06] and how to apply it with Attack Graphs.

Definition 1 (Balloon Drawings). *A balloon drawing is a graph or a tree where the layout of the node and the edges respects the following additional properties[LY06] :*

- **Property 1 :** *All children of a given parent are placed on a circle with the parent as the center.*
- **Property 2 :** *There are no edge crossings.*

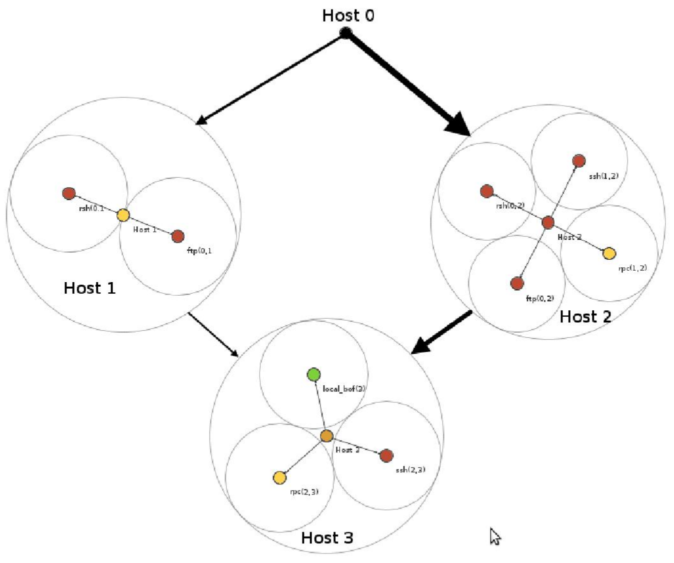


Figure 15: Example of a Balloon Attack Graph Using JUNG[OFWB03]

- **Property 3 :** *The deeper an edge is, the shorter its length.*

Discussion and Example

In Figure 15, we apply to our running example a clustering method with multiple cluster centers, one for each host or host group in the network, in order to form clusters of nodes without a pre-defined top-down path or a particular directional layout, based on the clustering methods proposed by Melancon et al. [MH98] and Lin et al. [LY06] which aim to achieve a balanced layout, namely, a *balloon attack graph.*

From the example, it is clear that this visualization model can improve the density of nodes as well as the readability to some extent, through clustering exploits associated to the same host. However, it is equally clear that the edges cannot be fully displayed (without breaking the balloons), leading to a significant loss of information; the improvement of scalability is also limited.

3.2.2 Attack Treemaps

An issue with conventional node-edge attack graph is the difficulty of expressing the hierarchical relationships between exploits, hosts, subnets, and networks. The above balloon attack graph addresses this through clustering nodes into balloons, but in doing so it also wastes much visualization space (to explicitly depict the hierarchical relationships).

To that end, treemaps allow for implicit representation of hierarchical information inside a rectangular display, where the entirety of the visualization space is put to use [JS91].

Attack Treemaps

Definition 1 (Network Attack Treemaps). *An Attack Treemap is a rectangle composed of a label (thin bar section on top) and a content pane (the remainder of the rectangle), in which the content pane is recursively divided into smaller Treemaps. It creates a compact two-dimensional representation of a tree-like structure, composed of vertical or horizontal recursive partitions of rectangles representing hierarchical relationships between different elements - exploits, assets or asset groups. The area of each partition is based on the weight assigned to each element, recursively defined as the sum of the weights of its children.*

At each level, every partition can be compared using both size and color, expressing two dimensions of data, two metric values for the element : one is partition size, the other is partition color. Treemaps appear as non-congruent tilings the plane, covering the plane without gaps or overlap, using similarly shaped tiles that vary with rotations, translations, reflections and scale.

Tiling Algorithms

The tiling algorithm defines how the sub-rectangles are partitioned inside their parent rectangle. Several tiling algorithms can be used for Treemaps, each satisfying different properties. They are generally a tradeoff between low aspect ratio (keeping the height of a rectangle as close to the width, instead of having a large discrepancy between these two values) and order. Table 4 compares the different Treemap tiling algorithms.

Algorithm	**Order**	**Ratio**	**Stability**
Slice-and-Dice	ordered	Very High	stable
Binary	ordered	High	stable
Squarified	unordered	Low	unstable
Strip	ordered	Medium	unstable

Table 4: Different Tiling Algorithms for Treemaps

The *slice-and-dice* tiling algorithm [JS91] is as follows : at every level, the partitioning direction is changed. The algorithm will simply alternate between horizontal and vertical partitioning of the rectangle. The *binary tiling* algorithm [JS91] dictates that whenever the width of the rectangle to be separated is greater than the height, the rectangle is to be partitioned vertically. If the height of the rectangle to be separated is greater than the width, the rectangle is to be partitioned horizontally. If the height of the rectangle is smaller than the width, this means that the rectangle will be separated vertically.

These two basic strategies may however lead to rectangles with extremely high aspect ratios, making the Treemap hard to analyze. Efforts have been made to reduce the aspect ratio of the subdivisions [BHW00, SW01]. They however are stable and are used in situations where order and stability are of great importance.

squarified treemaps [BHW00] propose to modify the tiling algorithm for treemaps to bring the aspect ratio of every subdivision as close to 1 (a square) as possible. The number of all possible tessellations is extremely large is considered NP-Hard. The authors however present an approximate approach based on recursively producing rectangles close to square shape for a set of siblings, without considering the subdivision for all levels at once. By sorting the input rectangles by size, the natural order of the rectangles is lost. The *strip treemap* [BSW02] is a simplified version of squarified as it works by laying out the rectangles in order in either horizontal or vertical strips, providing better readability than the default algorithm. While the squarified algorithm separates the rectangles in both directions, the strip Treemap algorithm only uses one, all the while maintaining order, at the cost of an increase in average aspect ratio.

Figure 2 describes a recursive function, which draws the Attack Treemap on a plane, using the *binary tiling algorithm*. The *binary tiling algorithm* [JS91] dictates that whenever the width of the rectangle to be separated is greater than the height, the rectangle is to be partitioned vertically. If the height of the rectangle to be separated is greater than the width, the rectangle is to be partitioned horizontally.

Algorithm 2: The Attack Treemap Algorithm

```
Input: An element $n$ with its position $p$, its height $h$ and its width $w$
Output: Calls a function which draws rectangular partitions of the appropriate size
        to satisfy the requirements of a Treemap
1  if $n$ has children then
2      Calculate the total sum of $n$, $totalSum = \sum_{i=0}^{m} score_{children}$;
3      for All children $c \in children$ do
4          Calculate the size ratio $r = \frac{score_c}{totalSum}$;
5          if $h \leq w$ then
6              for All children $c$ do
7                  $AttackTreemap(c, p, h, w * r)$
8          else
9              for All children $c$ do
10                 $AttackTreemap(c, p, h * r, w)$
11 Call $DrawElement(n, p, h, w)$ return;
```

The Treemap algorithm seen in Algorithm 2 does not take into account partition color. We use a color ramping algorithm described in Algorithm 1 in Chapter 3.1.

Discussion and Example

Figure 16 shows an *attack treemap* using our running example, built with the JavaScript InfoVis Toolkit [Bel] using the binary tiling algorithm [SW01]. In the attack treemap, each rectangle with a black bar at the top represents a host, inside which each colored rectangle represents an exploit. The color denotes the CVSS score, and the relative size of rectangles denotes the risk as defined in Chapter 3.1.

Clearly, treemap is dense and relatively scalable visualization model. In addition,

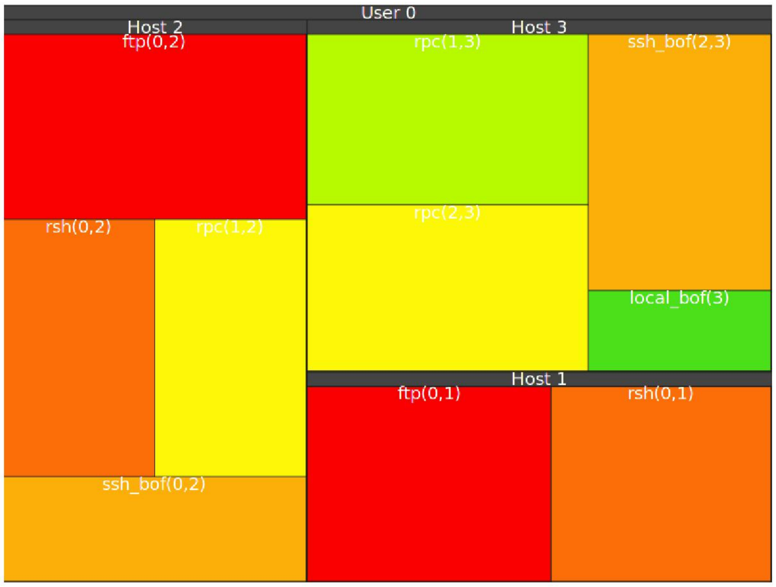


Figure 16: Treemap Example Using JIT[Bel]

GARNET[WLI08a] has shown how to add reachability results to treemaps by interactively displaying them through semantic substrates. However, most of the connectivity information and the edges in attack graphs are still missing in an attack treemap, and adding this connectivity information though overlying edges will clearly lead to a messy result. We will address this issue and enhance the treemap model in Chapter 4.

This visualization is suited for a high-level view of network host configuration, making it more useful for a systems administrator who has to manage system configurations and deploy patches rather than a security analyst who is seeking exploitation details or possible attack paths.

3.2.3 Hyperbolic Trees

As attack graphs get larger, screen size is a concern and a user must zoom on an area of the graph. This leads to a loss of context and awareness of the overall network. Hyperbolic geometry offers interesting features for scalability of large graphical structures. For any infinite straight line and any point not on it, there are many other infinitely extending straight lines that pass through and which do not intersect [And07]. This allows large amounts on non-crossing edges through the curving of lines. By creating a fisheye-lens effect, Hyperbolic geometry solves this problem by displaying the entire graph. The center of the graph - the zone of focus - occupies most of the space. The remainder of the graph is condensed and pushed back towards the outer edges, maintaining context and awareness of the whole visualization.

Network Hyperbolic Trees

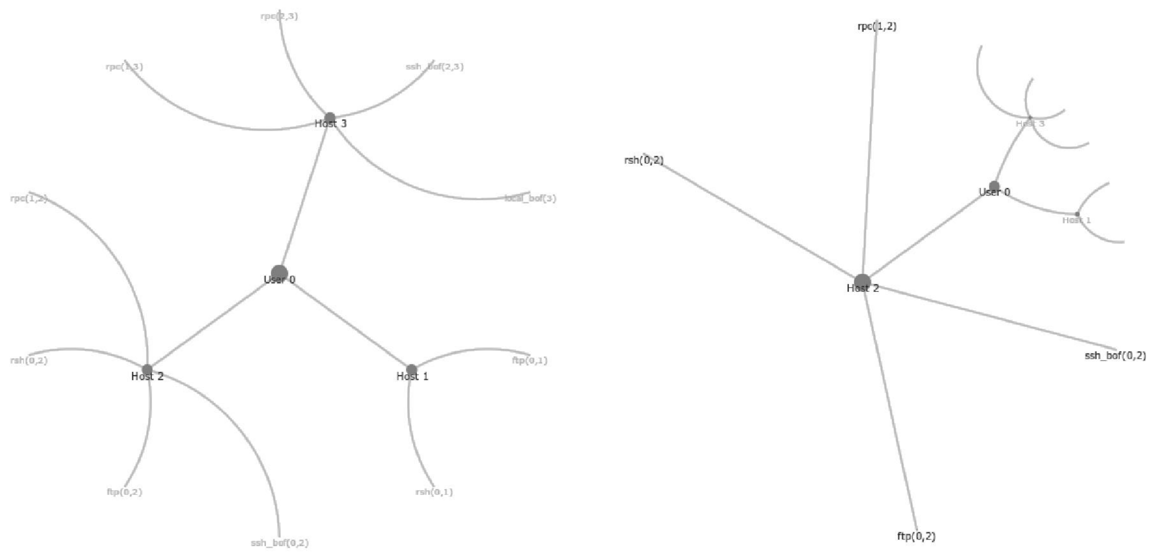
Here, we apply the concepts of hyperbolic geometry to the attack graph model. We will use the implicit hierarchy $Exploit \in Host \in Subnet$ to specify the multiple levels of our tree.

Definition 1 (Hyperbolic Trees). *Hyperbolic trees are euclidian trees transposed on a hyperbolic plane. For an attack graph AG we take all the exploit nodes $e \in Exploits$, all the host nodes $h \in Hosts$ and all the host group nodes $g \in Groups$ and display this hierarchy : If an element at one belongs to a an element at the level above, an edge is drawn between these two elements. A Tree $T(N,E)$ where N is a set of nodes representing exploits or user-access conditions and E a set of edges showing the relationship between nodes.*

Discussion and Example

Figure 17 shows an example of a hyperbolic tree based on the attack graph in Figure 2. Subfigure 17a is centered on the attacker's initial user condition *user*0. Subfigure 17b is re-centered on the user-access condition on host 2, *user*2.

As hyperbolic trees cannot model complex connections - a node can only have a single parent - they are more suited for physical connectivity graphs of networks rather than attack graphs. However, the constant contextual awareness makes hyperbolic attack trees an



(a) A Hyperbolic Tree (b) A Re-Centered Tree

Figure 17: A Hyperbolic Tree using JIT [Bel]

appealing choice for applications like situational awareness. We will further enhance this approach with additional features in Chapter 5.

3.2.4 Limitations of Existing Visualizations

While conventional attack graph visualizations are not scalable, it appears that existing visualization methods are not well suited to the display of attack graphs. Because of their implicit hierarchy - an exploit using a vulnerability which is on a host - and their high connectivity - an exploit can be executed from multiple hosts - and because multiple conditions are required to satisfy an exploits preconditions, it is very difficult to visualize attack graphs using existing paradigms.

While some efforts have been made to cluster graphs [NJ04], we believe that higher levels of abstractions are needed, and distancing ourselves from the node-edge paradigm would be preferable to ensure proper scalability of attack graph visualizations. This exercise has, however, allowed us to extract positive features from some of these visualizations, and two characteristics overwhelmingly stand out in these experiments :

	Scalability	**Connectivity**	**Density**
Node-Edge Graphs	Low	Full	Very Low
Balloon Graphs	Low	Partial	Medium
Treemaps	Very High	Low	Very High
Hyperbolic Trees	High	Partial	Medium

Table 5: Limitation of Existing Visualizations

- Treemaps offer very high information density and are extremely scalable, at the cost of connectivity. We will address these shortcomings in Chapter 4, where we will introduce a treemap-based visualization scheme for network overview.

- Hyperbolic trees offer good scalability and they allow the user to constantly maintain contextual awareness. We will use these characteristics to drive a visualization model suited for cyber-situational awareness in Chapter 5, where we will present a tree-based scheme suited for real-time intrusion detection systems using attack graphs.

Table 5 shows an overview of the limitations of the existing visualizations reviewed in this chapter.

Chapter 4

Radial Attack Treemaps

This chapter introduces a scalable, metric-driven visualization model, the *radial attack treemap*, for the purpose of obtaining a quick overview of a network's vulnerability information. We first give an overview, followed by the description of models and algorithms, and finally we present simulation results.

4.1 Overview

Enabling a security analyst to acquire a detailed overview of the entire network is a key tactical advantage in assessing a network's security. The goal here is to encode as many legible details as possible, in a single view, inside a given canvas size. Chapter 3.2.2 mentionned treemaps as a visualization model providing relatively high information density and scalability by occupying the entirety of the canvas. On the other hand, the main shortcomings of treemaps lie in the difficulty of displaying the edges and relationships between different exploits in the network.

Intuitively speaking, our main idea is to *bend the treemap into a ring, and display the edges in the center of this ring*. As for the actual display of the edges, we turn to radial graphs, which allow a fixed-size layout with high information density, element proximity and edge management [KSB$^+$09, LAM$^+$05]. Unlike conventional graphs in which edges may be obstructed by a node, a distinguishing property of radial graphs is that a line between two points on can be a straight, unobstructed line. Moreover, the edges in a radial graph can be hierarchically bundled [Hol06] with crossing between edges minimized [HN10].

By combining key concepts of treemaps and radial graphs, we propose a metric-driven and treemap-based radial visualization, namely, the *radial attack treemap*. We summarize the key features and advantages of this novel visualization model in the following, while leaving details of the model and implementation to later sub-sections:

- The model provides a quick overview of exploits, chains of exploits (that is, paths in an attack graph), hosts, and causal relationships between exploits in a network.
- The color and size of each slice of the outside ring represents the CVSS score and risk of the corresponding exploit, respectively.
- The stacking of slices and sub-slices in the outside ring implicitly represent hierarchical relationships between exploits, exploit chains, and hosts, reducing the number of edges that need to be explicitly displayed (in contrast to the original attack graph).
- The center of the ring displays edges in a bundled way to minimize the number of crossings between edges, leading to a cleaner visualization result.
- Layout of the bent treemaps is optimized such that the lower level details are displayed more towards the outer side of the ring in order to occupy more space.

Figure 18 is a mockup illustrating the ideas behind the radial attack treemap, and how the different elements and their relationships are transcribed to the canvas.

4.2 Models and Algorithms

Definition 4.2 more precisely describes the radial attack treemap (additional details are still left to later sections).

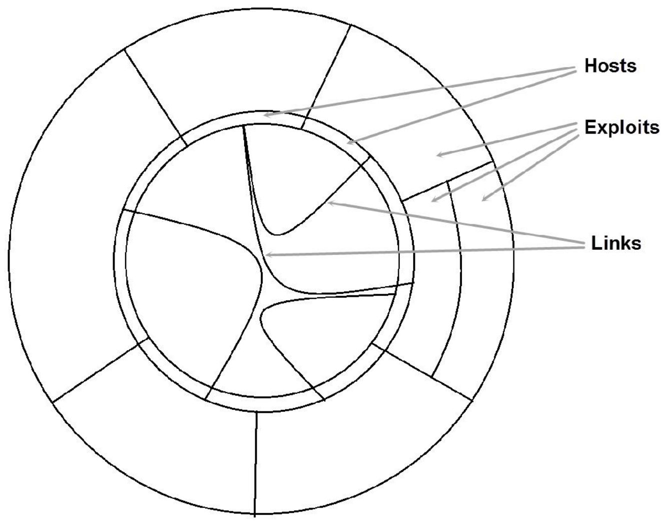


Figure 18: A Mockup of the Radial Attack Treemap

Definition 1 (Radial Attack Treemap). *Given an attack graph $G(E \cup C, R_r \cup R_i)$ with hosts H, the asset value assignment function AV, and the risk function R_c, R_h, and R_g, a radial attack treemap is composed of a ring R and a collection of links L, where*

- *R is divided into a collection of slices S, with each slice $s \in S$ corresponding to a host $h \in H$.*
- *each slice s is divided into a collection of subslices SS, with each subslice $ss \in SS$ corresponding to an exploit chain (a sequence of exploits involving the destination host h, the same source host, and leading to $< root, h >$.*
- *each subslice ss is further divided into a collection of subsubslices SSS, in which each subsubslice $sss \in SSS$ corresponds to an exploit e in the exploit chain.*
- *the relative size of each slice, subslice, and subsubslice is proportional to the risk score (Definition 3.1.1) of corresponding host, exploit chain, and exploit, respectively (details will be provided later).*
- *the color of each subsubslice represents the CVSS score of the corresponding exploit (details will be provided later).*

- *each link in L points from a slice corresponding to host h, to a subslice corresponding to an exploit chain involving the source host h.*

- *all the links in L are bundled and routed through the center of the ring R.*

Figure 19 illustrates an example of radial attack treemap, which is based on our running example shown in Figure 2.

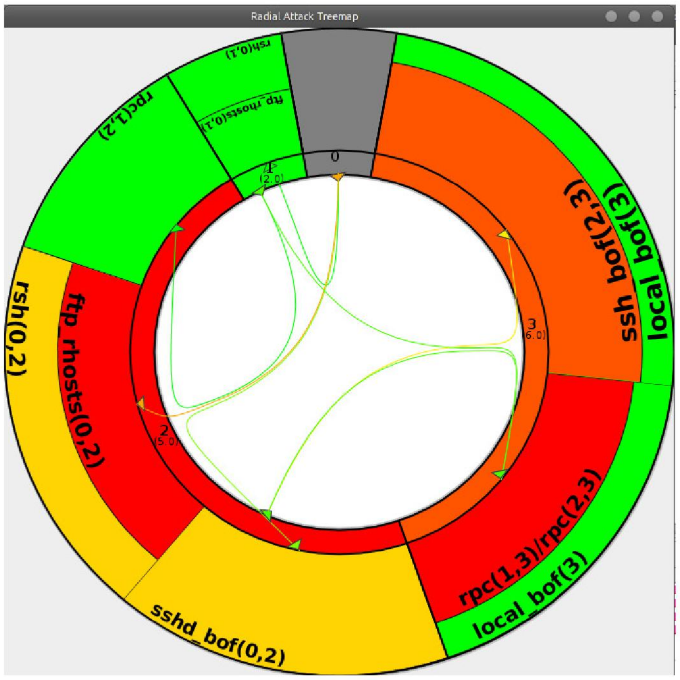


Figure 19: Radial Attack Treemap representing Figure 19

4.2.1 Data Structures

We now describe the data structures required for implementing the proposed visualization model. Specifically, to implement the model, we need to compute the aforementioned risk metrics and convert a given attack graph into a suitable data structure. We then derive geometric information necessary to the final rendering of the model. Therefore, for each element in the model, there will be a corresponding view element containing additional information necessary to the visualization, as detailed below.

- *Exploit & Subsubslice:* Each exploit is a list of five attributes: an identifier, a set of pre-conditions and post-conditions, a CVSS score, and a risk value. Correspondingly, a subsubslice, as the view representation of the exploit, is a list of attributes including a label, a color derived from a normalized CVSS score, as well as a size proportional to the risk value of exploit chain.

- *Exploit Chain & Subslice:* Each exploit chain is a list of attributes including an identifier, a risk value, as well as the source host involved. Correspondingly, a subslice is a list of attributes including references to the composing subsubslices, a label, a size derived from the risk value, an anchor point which is a set of coordinates used as destination points for incoming links, and a color derived from the CVSS scores of the corresponding exploits.

- *Host & Slice:* A host is a list of attributes including the references to the composing exploit chains, an identifier, and a risk value. Correspondingly, a slice is a list of attributes including the host name, references to the composing subslices, a label, a color derived from the CVSS scores, a size derived from the risk value of the host, and two anchor points, with the first being a set of coordinates used as intermediate destination points for incoming links and the second being a set of coordinates used as the source points for outgoing links from this host.

- *Link:* A link is a pair $< h, ec >$ indicating the source host h invovled by exploits in the exploit chain ec. Correspondingly, the link is visulized using the Bézier spline composed of two curves, a cubic Bézier curve and a quadratic Bézier curve [PBP02]. The former contains three sets of coordinates, namely, a start point, an end point and a control point, while the latter has four, namely, a start point, an end point and two control points.

4.2.2 Algorithms

This subsection discusses two series of algorithms. The first converts a given attack graph to the data structures mentioned in the previous sub-section. The second is for computing geometric information used in creating the view structures.

First, in the following, Algorithm 3 uses a recursive depth-first search in the input attack graph to obtain all paths from user-access conditions to the root condition of the target host (Algorithm 4). For each path obtained, we verify that all exploit sequences leading to this condition have all their pre-conditions satisfied and that the path generated is valid (Algorithm 5).

Algorithm 3: GETALLEXPLOITCHAINS

```
Input: An attack graph, a set of host-access conditions $Host$
Output: A set of Hosts possessing exploit chains and exploits
1 foreach $Host\ to \in Hosts$ do
2     foreach $Host\ from \in Hosts$ do
3         $paths_{from->to}[\ ][\ ] \leftarrow getAllPaths(from, to)$;
4         foreach $path\ p \in paths_{from->to}$ do
5             if $isValid(true, path, from, initialconditions)$ then
6                 $to.addExploitChain(path)$;
```

Algorithm 4: GETALLPATHS

```
Input: A Linked List of visited nodes $visited$, the end condition $end$
1 $Node\ n = visited.last()$;
2 $Node[]\ nodes = n.getNexts()$;
3 foreach $Node\ n \in Nodes$ do
4     if $visited.contains(node)$ then
5         $continue$;
6     $visited.add(n)$;
7     $Node[]\ path \leftarrow visited$;
8     $allPaths.add(path)$;
9     $visited.removeLast()$;
10 foreach $Node\ n \in Nodes$ do
11     if $visited.contains(n)\ ||\ n = end$ then
12         $continue$;
13     $visited.addLast(n)$;
14     $getPath(visited, end)$;
15     $visited.removeLast()$;
```

```
Algorithm 5: isValid
  Input: A boolean condition indicating the validity of the path $isValid$, a list of nodes
         path $path$, a starting node $start$ and a list of satisfied conditions
         $satisfiedConditions$
  Output: Recursively checks if all exploits have their preconditions satisfied
1 $satisfiedConditions.add(from)$;
2 $satisfiedConditions.add(from.getPreconditions())$;
3 foreach $Node\ next : from.getNexts()$ do
4     if $next.getPreconditions() \in satisfiedConditions$ then
5         return $isValid(true, path, next, satisfiedConditions)$;
6     else
7         return $false$;
```

Second, we discuss how the view data structures may be generated. The ring is generated by converting exploits, exploit chains and hosts into subsubslices, subslices and slices, respectively. Host and exploit chain risk scores are expressed by the angle of ring segments they occupy. $Host_0$, representing the initial attacker-controlled host, possesses a fixed angle, α_0. The slices representing a given host x will have an angle α_x of value :

$$\alpha_x = ((360 - \alpha_0) * \frac{score_x}{\sum_{i=1}^{n} score_i}) \tag{5}$$

Similarly, the angle α_y of an exploit chain $ec \in h_x$, relative to risk of the other exploit chains of the host - will have a value of :

$$\alpha_y = \alpha_x * \frac{score_{ec_i}}{\sum_{ec \in x} score_{ec}} \tag{6}$$

For an exploit $e \in ec$, the angle of ring segment it occupies is the same as that by its exploit chain parent, and occupied area thus depends on the length of the radius segment between the current exploit and the next exploit (or the ring's two edges), depending on the risk scores of these exploits' post-conditions.

The color of subsubslice is derived from the normalized CVSS scores of the vulnerabilities using a *color ramping algorithm* similar to the one described by Bourke in [Bou], described in Chapter 3.1, Algorithm 1.

The links displayed at the center are Bézier splines [PBP02] computed using the method by Holten in [Hol06]. The spline is composed of two Bézier curves, the *origin curve* and the *destination curve*, as detailed below and illustrated in Figure 20 (where the numbers indicate the starting, end, and control points, and the red tree shows the hierarchical structure for edge bundling). Finally, the link color is derived from the score of the first exploit in the exploit chain of the destination, using a color ramping algorithm.

- The *origin curve*, a cubic Bézier curve (with two control points), starts at the origin host's anchor point denoted by point 0 and ends on the destination host's projection on the host circle, point 3. The control points for this curve are the host origin and destination points, respectively projected on the inner host circle, denoted by points 1 and 2, respectively.
- The *destination curve* is a quadratic Bézier curve starting at the end of the origin curve, point 3, and ends at the exploit chain anchor point, point 5. The control point is the projection of the host destination point on the host label ring, point 4, allowing us to properly separate the different links leading to different exploit chains on the same host.

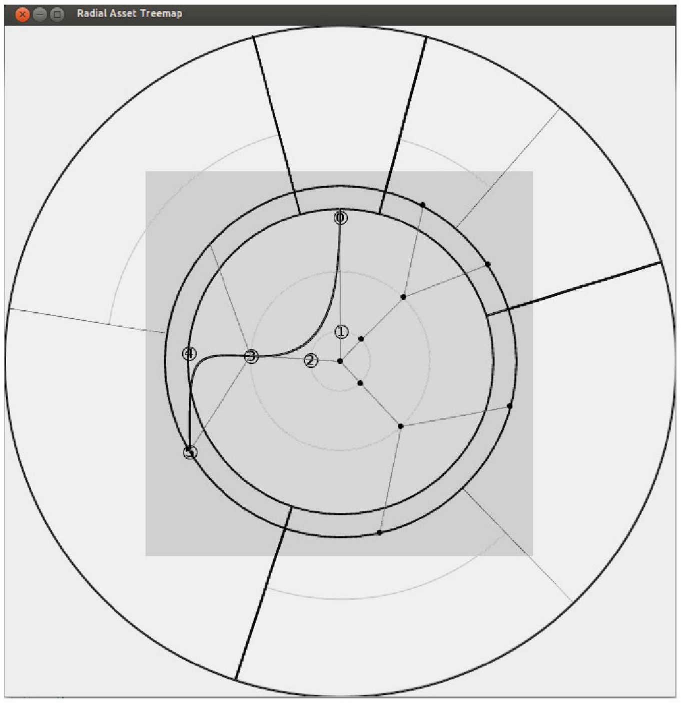
4.3 Implementation and Simulation

In this section, we share techinical details retagarding implementation as well as the setup, result, and analysis of our simulations.

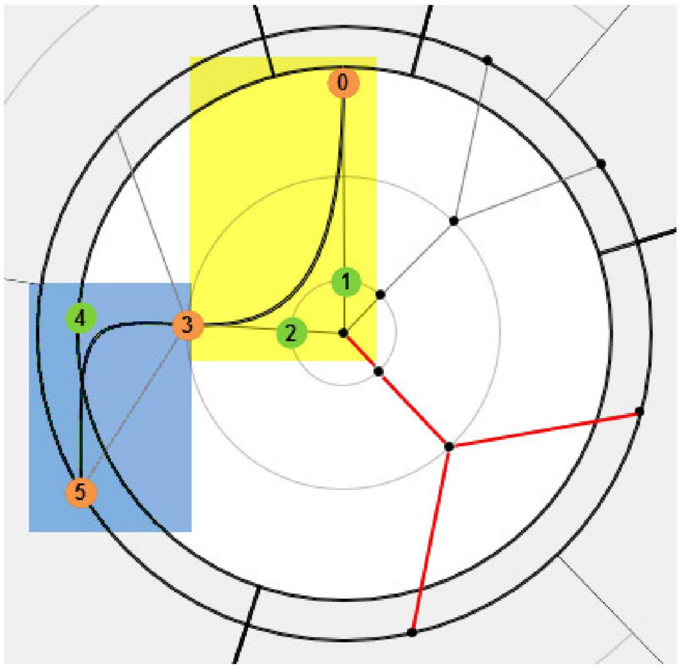
4.3.1 Implementation Details

A prototype was built using Java and the Graphics2D and Curve2D libraries, included in the JavaSE package. It is built using the Model-View Controller [KP+88] (MVC) pattern. A GraphViz[EGK+01] *.dot* file parser reads an input attack graph and loads it into memory. The graph is then traversed to generate exploits, exploit chains, and hosts, using Algorithms 3,4 and 5. This model is then converted into slices, subslices, subsubslices, and links, in order to generate the ring and links.

Figure 21 illustrates a larger attack graph and corresponding radial attack treemap.

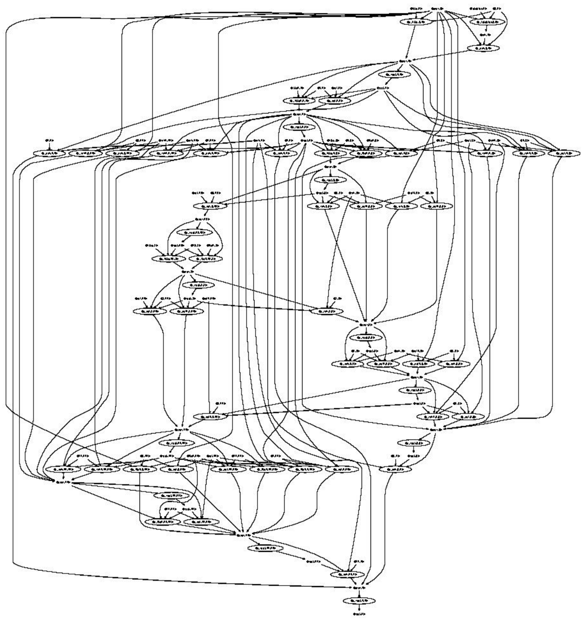


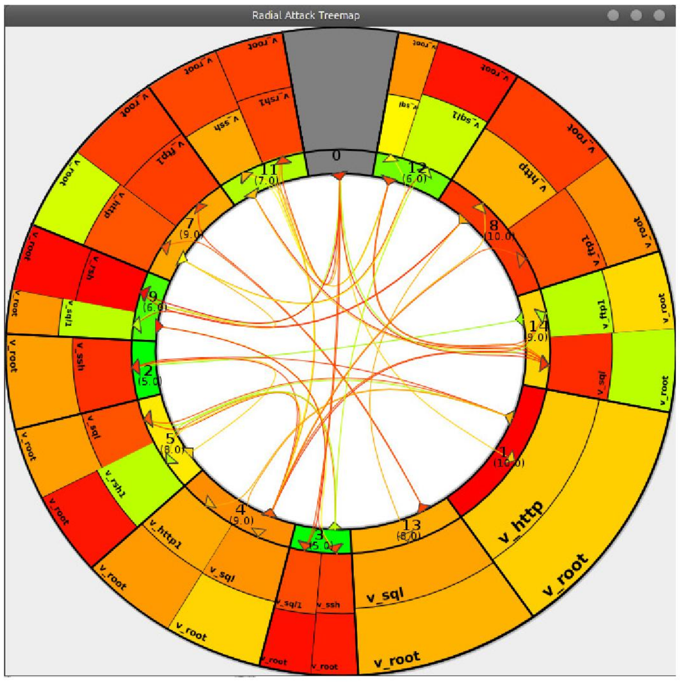
(a) Overall of the Center of the Visualization



(b) Zoomed-in on the Splines

Figure 20: Link Generation


(a) The Attack Graph to Visualize



(b) The Corresponding Radial Attack Treemap

Figure 21: A Larger Example of a Radial Attack Treemaps

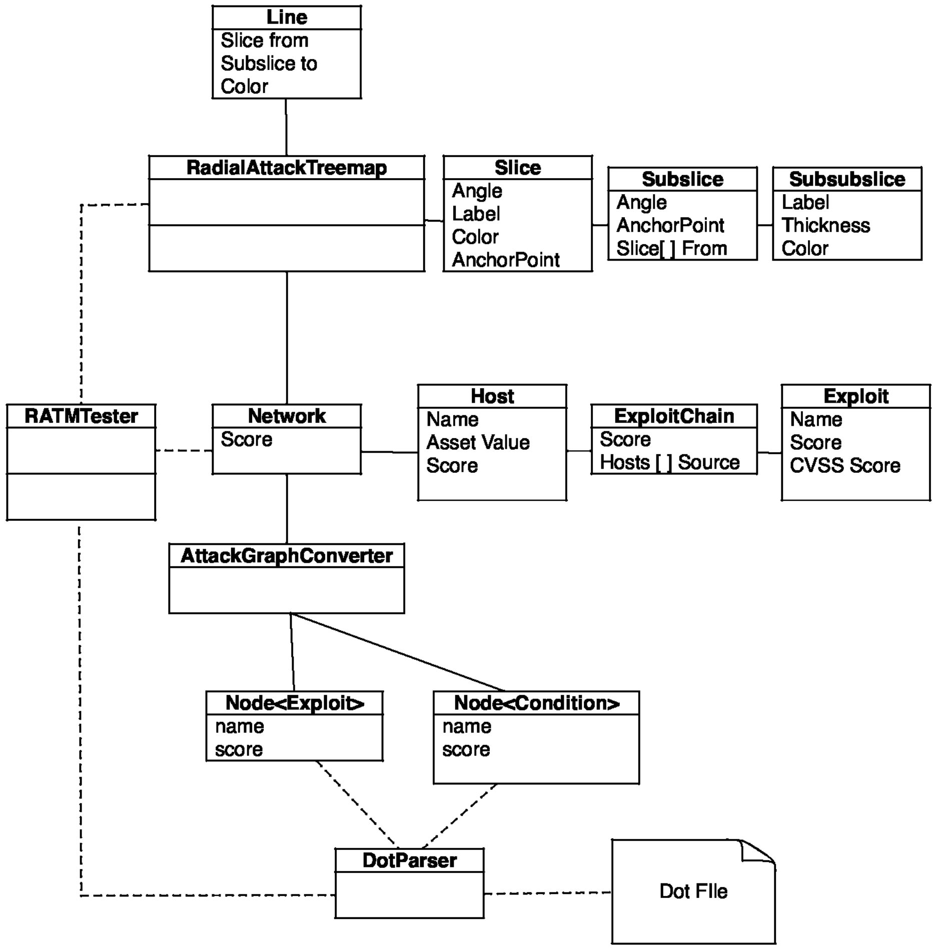


Figure 22: Simulation Software Setup

4.3.2 Simulation Setup

We now study the density and scalability of the visualization model through simulation using randomly generated attack graphs (we note that although an experiment using real world data is certainly more desirable, to the best of our knowledge, a publicly available dataset containing a significant number of attack graphs is not currently available). We generate 1200 attack graphs using a Python application from small seed graphs based on real world attack graphs. The simulation environment is a dual-core Intel Core i5 processor with 8GB of RAM running Debian 7. The entire application was written in Java and runs on OpenJDK 6. Figure 22 shows an overview of the overall architecture of the system.

Information on the average number of node and edges of these graphs is illustrated in

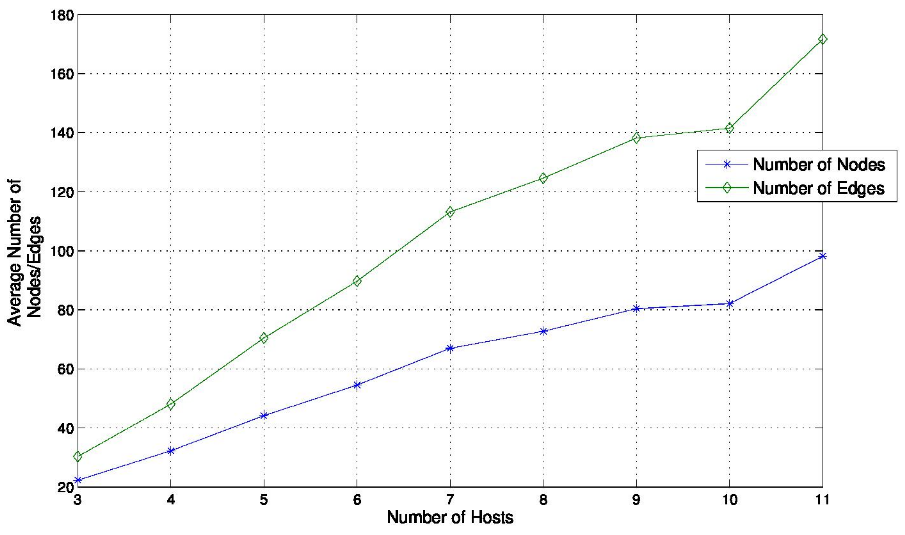


Figure 23: Average of Edges and Nodes per Host

Figure 23.

We compare the scalability of radial attack treemaps with that of the input attack graphs. As a radial attack treemap is designed as a fixed-size visualization, we set a threshold value for the smallest allowable subsubslice, at $1000px^2$ (leaving approximately 10 characters at 8pt. font size), and we ensure all subsubslices in a radial attack treemap to be legible by scaling them according to this threshold. Figure 24 shows the average canvas size of both models in relation to the number of hosts.

4.3.3 Simulation Results

The simulation clearly confirms that radial attack treemaps offer a higher information density than conventional attack graphs. Figure 24 shows that, on average, 10 hosts can be represented on a canvas of merely 900x900 pixels, while the corresponding attack graphs would require a canvas of over 2800x2800 pixels.

Next, we study the degree of reduction in the number of edges/links by implicitly representing edges in the radial attack treemaps (through stacking subsubslices). We note that, in addition to this reduction in the number of edges/links, the radial attack tree maps have

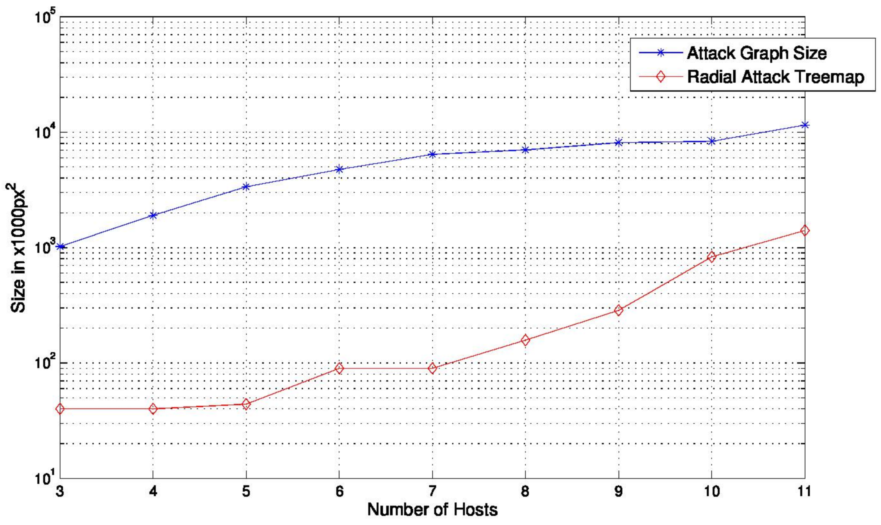


Figure 24: Average Size of Attack Graphs and Radial Attack Treemaps

other advantages in terms of displaying links, as mentioned already in the previous subsection. Figure 25 compares the number of edges/links in relation to the number of hosts in the graph.

This simulation indicates that the amount of edges/links has been reduced to approximately a third those of conventional attack graphs. The implicit relationships between host, exploit chains and exploits allow for such a significant edge reduction.

We note that, Figure 24 seems to indicate that the rate of growth of radial attack treemap is greater than that of conventional attack graphs. This is a consequence of the ring's tiling algorithm: regardless of the size of the canvas, an exploit chain's partition angle will remain the same. When compared to a two-dimensional graph canvas, both size increases are quadratic but with the partition size depending on the angle of it's parent exploit chain, leading to a lower rate of growth of partition surface compared to the available surface of a rectangular canvas.

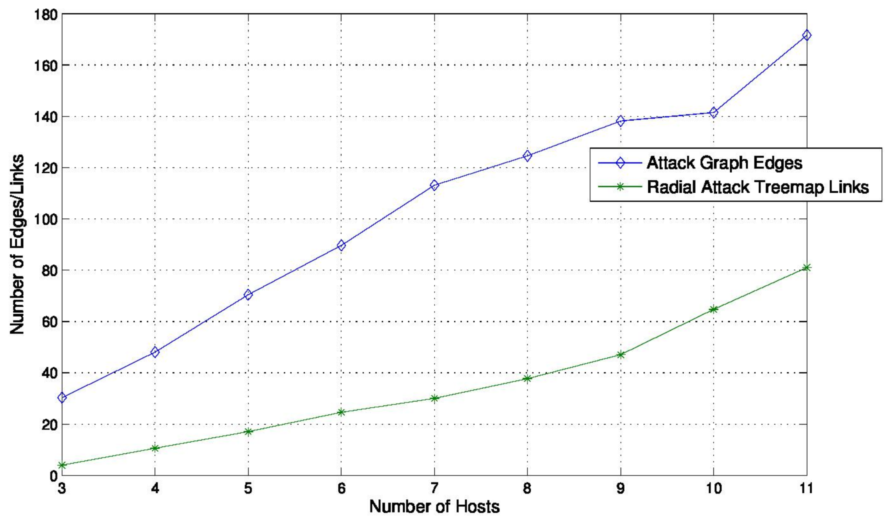


Figure 25: Average Number of Edges and Links

4.4 Discussions and Limitations

The proposed metric-driven radial attack treemap provides a viable visualization solution for human analysts, allowing to observe the entire network at a glance in a single view with vulnerability information, host configuration information, vulnerability metrics as well as connectivity information. The representation of two metrics through both size and color, allow the analyst to easily spot outlying exploits. We have shown through automated simulations based on randomly-generated attack graphs that this model offers superior information density when compared to conventional attack graphs.

Nonetheless, the model in its current form still has a few limitations. First, due to the limited level of hierarchy in the treemaps, it will be difficult to visualize a large network in a single view, without resorting to zooming and scrolling. Developing a new tiling algorithm to support more levels of hierarchy is one potential solution. Another solution would be the use of interactivity, by varying the hierarchy levels of a slice based on user focus, or by filtering out certain hosts, exploit chains, and exploits. Second, the trade-off between the areas occupied by the ring and by links requires further study. Algorithms that can optimize this trade-off in order to ensure the best clarify for both sides needs to be developed. Finally, algorithms are also needed for the efficient incremental updates of the model when networks or vulnerabilities change.

Chapter 5

Topographic Hyperbolic Trees

This section introduces the novel *topographic hyperbolic tree* model for monitoring and predicting real time progress of attacks. We first give an overview, followed by the description of models and algorithms, and finally we present simulation results.

5.1 Overview

One important aspect of visualization in the application of cyber-situational awareness is to allow administrators to see both the current focus of an ongoing attack and the most likely next steps. Another important aspect is to provide a sense of *distance* between potential attack steps based on the number of intermediate steps or relative difficulty of such steps [WLI08a, WLI08b]. In Section 3.2.3, we have shown that the hyperbolic tree model is a suitable model for the first purpose.

As to express the attack distance, we are inspired by geographical topographic maps, in which contour lines are used to indicate fixed increases in altitude. Therefore, the main idea here is to *enhance the hyperbolic attack tree model with contour lines representing attack steps at similar distance.*

Again, we summarize the key features and advantages of this novel visualization model in the following, while leaving details of the model and implementation to later sections:

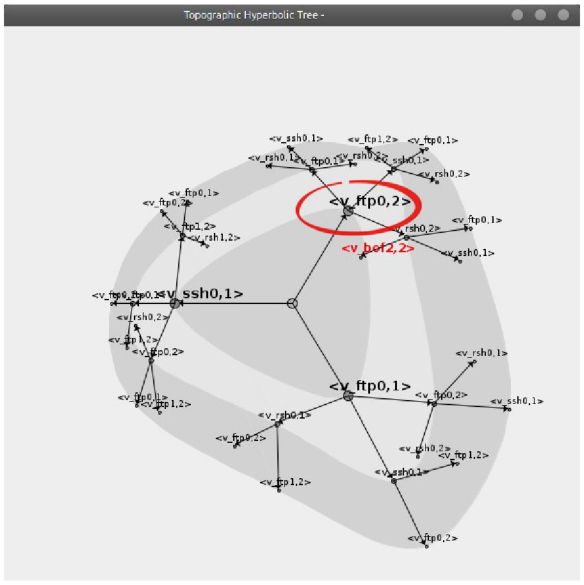


Figure 26: Topographic Hyperbolic Tree

- The model provides an interactive visualization of the ongoing attack and its plausible next steps.

- The hyperbolic tree creates a fisheye-lens effect that allows administrators to focus on the current attack and its closest future steps, while not losing context or awareness of other steps that may be further away but are still possible, such as the ultimate goal of the attack.

- The contour lines provide a rough idea about future attack steps that are at similar distance from the current step.

- In addition, the relative length of different edges represent (after taking into account the fisheye-lens effect) the relative difficulty of the corresponding exploit.

Figure 26 illustrates an example of topographic hyperbolic tree based on our running example.

5.2 Model and Algorithms

Definition 5.2 formally describes the topographic hyperbolic tree representing an attack graph.

Definition 1. *Given an attack graph $G(E \cup C, R_r \cup R_i)$ with hosts H and the risk function R_c, R_h, and R_g, a topographic hyperbolic tree is composed of a hyperbolic attack tree $T(E,C)$, which has the exploits E as nodes and conditions C as edges, and a collection of contour lines L linking all the exploits sharing the same depth in the tree. The relative length of an edge is based on the risk metric score of the corresponding condition as well as the depth of the node.*

5.2.1 Data Structures

We now describe the data structures required for implementing the proposed visualization model. Specifically, to implement the model, we need to compute all possible attacker paths and form a tree of the different sequence of exploits the attacker may perform. We then derive geometric information necessary to the display of the visualization.

To elaborate the tree, we need the following structures:

- **Exploit:** An exploit $e \in E$ contains a name, a CVSS score as well as two sets of conditions: the pre-conditions and the post-conditions.
- **Condition:** A condition $c \in C$ contains a name as well as the condition risk, as defined in 3.1.
- **Attacker Knowledge:** The attacker knowledge AK is the set of conditions $c \in C$ which have been obtained by the attacker. If the pre-conditions of an exploit $e \in E$ are a subset of the attacker knowledge ($pre(C) \subset AK$) the exploit e may be executed. Once e is executed, the post-conditions of e will be added the attacker knowledge: $AK = AK \cap post(e)$
- **Attack Tree:** An attack tree is a set of all possible exploit paths an attacker can take in order to achieve his goal - the end condition of the attack graph. It is represented

by a tree $T(E,C)$ with exploits E as nodes and a set of post-conditions C as edges. Section 5.2.2 will explain how to obtain this structure from an attack graph.

Once the tree is constructed, it is displayed on screen using the following structures:

- **Node:** A set of nodes $n \in N$ will form a tree $T(N)$, each node n possessing a size and an angle, as well as a list of it's children nodes.
- **Edge:** An set of edge $e \in E$ will link a pair of nodes $< n,n >$ and will have as unique attribute its length.
- **Ring:** The ring is used to draw the topographic lines. A ring $r \in R$ is a set of points $p(x,y)$ representing all possible exploits at a given step.

Additional details on how these structures are used will be explained in Section 5.2.2.

5.2.2 Algorithms

The construction of a topographic hyperbolic tree from an input attack graph involves a few steps. We first load the attack graph into memory. Then, for each time the graph is re-centered, we apply the tree generation algorithms. We then layout the tree on the canvas and generate the contour lines. More specifically,

1. We start by establishing the context required to initiate the graph traversal using (Algorithm 6), and then recursively perform the graph traversal and tree construction using Algorithm 7, while limiting the maximal depth of any tree branch to be a pre-defined parameter MAX_DEPTH in order to avoid the explosion of possible paths.
2. We then layout nodes on the canvas. We compute the coordinates of every node by calculating the length of a link as well as the angle at the origin. The length of an edge is a function of the risk of the pre-conditions of the exploit represented, as well as the number of steps from the center:

$$distance_{child} = \frac{score_{child}}{c} * (MAX\_DEPTH - step_{child} + 1)$$

. The angle of a node's children will depend on the angle of the parent as well as the number of children this parent possesses:

$$\alpha_c = \frac{180 - c * step}{nbChildren}$$

.

3. Finally, we generate and draw the contour lines. Three main steps are required for the drawing. First, after obtaining all points at a given level i, we ensure that the polygon formed by these points completely includes the polygon formed by the points at a previous level $i-1$. Otherwise, we add the points of polygon $i-1$ lying outside of polygon i to the polygon i. Second, we ensure that each polygon is convex. If the polygon is concave, we apply a convex-hull algorithm commonly called the Gift-Wrapping Algorithm [Jar73]. Finally, we smooth the lines by interpolating the points using the Catmull-Rom Algorithm [CR74].

Algorithm 6: THE TRAVERSAL INITIATION FUNCTION

```
Input: A tree $tree$, an attack graph $graph$, a list of conditions $attackerKnowledge$
Output: A tree $tree$ representing all possible attacker paths from given initial
        conditions
1 $attackerKnowledge.add(initialConditions)$
  $Node[\,]\ firstNextSteps \leftarrow getNextSteps(attackerKnowledge)$;
2 for $Node\ n \in firstNextSteps$ do
3     $attackerKnowledge.add(n)$;
4     $attackerKnowledge.add(n.getNexts())$;
5     $traverse(tree, n, attackerKnowledge, 1)$;
6 return $tree$;
```

5.3 Implementation and Simulation

In this section, we share details of the implementation as well as the setup, result, and analysis of our simulations.

Algorithm 7: THE TREE GENERATING ALGORITHM

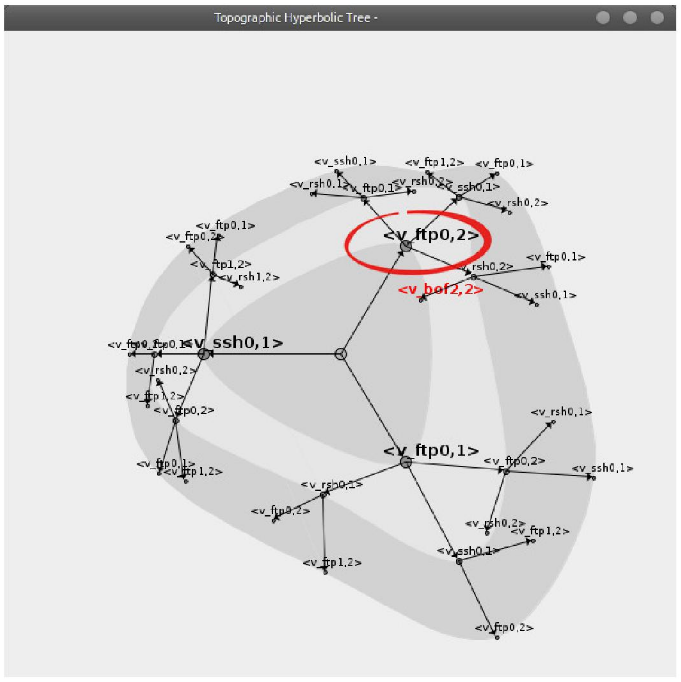
```
Input: A tree node $previous$, an attack graph node $graphNode$, a list of conditions
       $attackerKnowledge$ a depth $depth$
Output: The fully expanded tree representing all possible attacker paths
1  $current \leftarrow graphNode$;
2  $previous.addNext(current)$;
3  $current.addPrevious(previous)$;
4  if $previous \neq FINAL\_CONDITION$ && $depth \leq MAX\_DEPTH$ then
5      $Node[\,]\ nextSteps = getNextSteps(attackerKnowledge)$;
6      for $Node\ n \in nextSteps$ do
7          if (! $attackerKnowledge.contains(n.getNexts())$) then
8              $attackerKnowledge.add(n.getNext)$;
9              $attackerKnowledge.add(n)$;
10             return $traverse(current, n, attackerKnowledge, depth + 1)$
```

5.3.1 Implementation Details

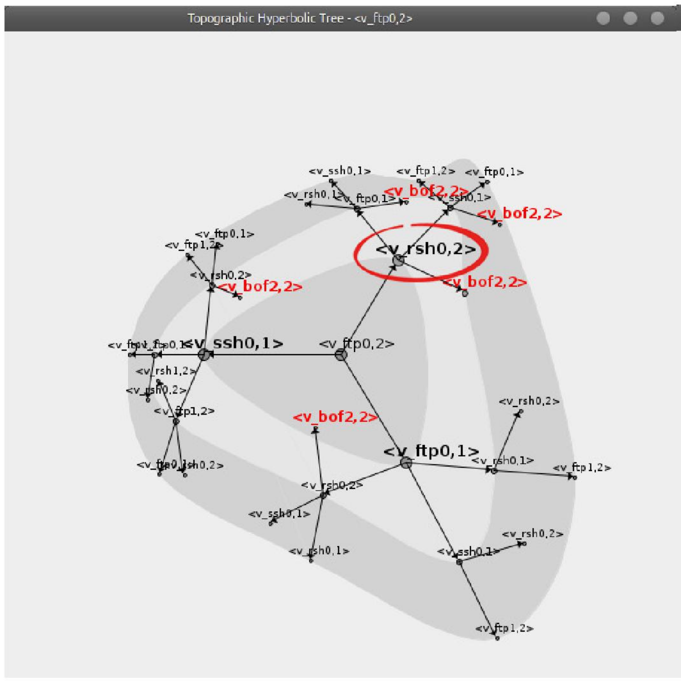
Just like in Chapter 4, the prototype was built using Java and the Graphics2D library, using the MVC[KP$^+$88] pattern, parsing GraphViz .dot files to automatically generate the tree. We illustrate the network and attack graph depicted in Figure 2 from Chapter 2.1 using our newly introduced technique. The results are illustrated in Figure 27 where we re-center the visualization twice. Figure 27a is the visualization centered on the initial conditions, the entirety of the attacker's knowledge. Figure 27b is then re-centered on the exploit $v\_ftp0,2$, the attacker has now gained the condition $trust0,1$, and may execute the previously available exploits and $v\_rsh0,2$ due to newly-acquired knowledge. Once the attacker has exploited $v\_rsh0,2$, he has acquired user access on host 2, and Figure 27c depicts the attacker's potential next steps. The attacker is now one step away from his goal condition, indicated in red, $v\_bof2,2$.

5.3.2 Simulation Setup

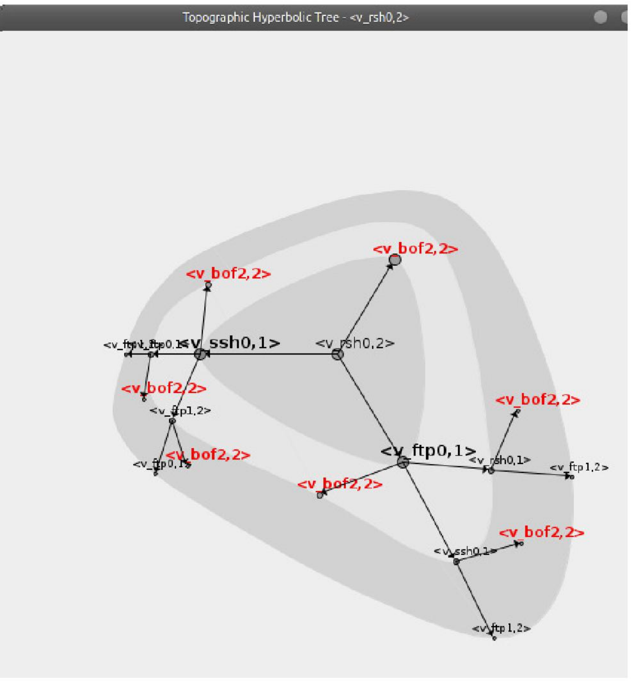
Simulation environment is identical to the one in Chapter 4. In this experiment, we use 3500 randomly generated attack graphs. Figure 28 shows an overview of the application's architecture.



(a) Centered on the Initial Condition



(b) Centered on the Conditions Brought by $v\_ftp0,2$



(c) Centered on the Conditions Brought by $v\_rsh0,2$

Figure 27: Example of a Topographic Hyperbolic Tree With Varying Centers

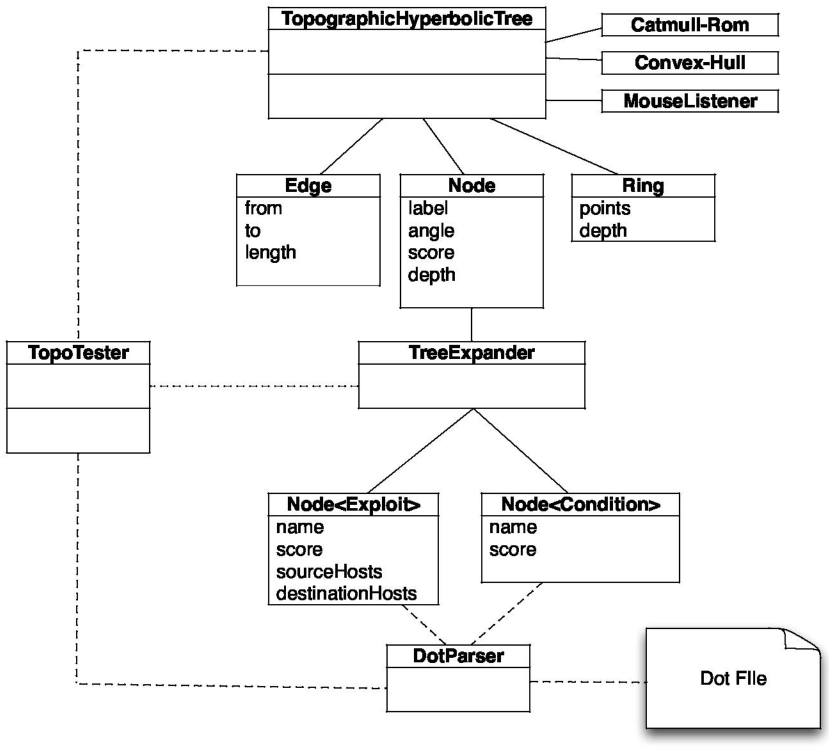


Figure 28: Simulation Software Setup

5.3.3 Simulation Results

Figure 29 analyzes the average number of nodes in a tree as a function of maximum tree depth for different amounts of hosts (logarithmic scale in Figure 29a and a linear scale in Figure 29b). The latter allows us to more easily grasp the repercussions of tree depth.

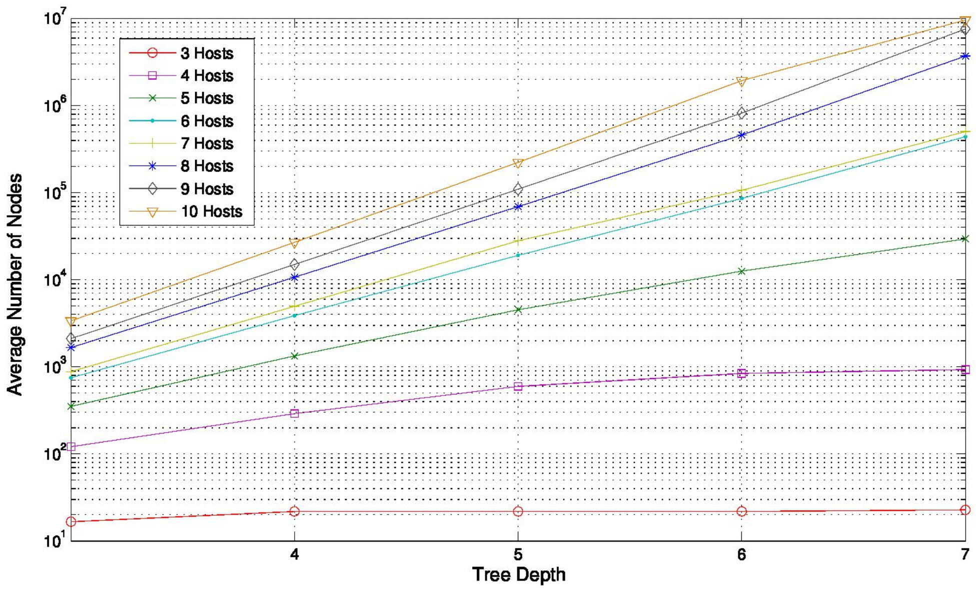
Intuitively, the tree expansion algorithm's computational complexity should increase in a very similar fashion to the amount of nodes in the tree. Figure 30 confirms this hypothesis by presenting average computational complexity of the tree expansion algorithm as a function of tree depth for different network sizes.

From the simulation results obtained, it is clear that the implementation of a maximum tree depth is necessary ; there is a very sharp increase in the amount of nodes - and intuitively, time - following the sixth step. Unsurprisingly, there is an exponential increase in nodes, despite the monotonicity assumption. Our simulation results indicate that limiting the maximal tree depth to five or six should easily be manageable.

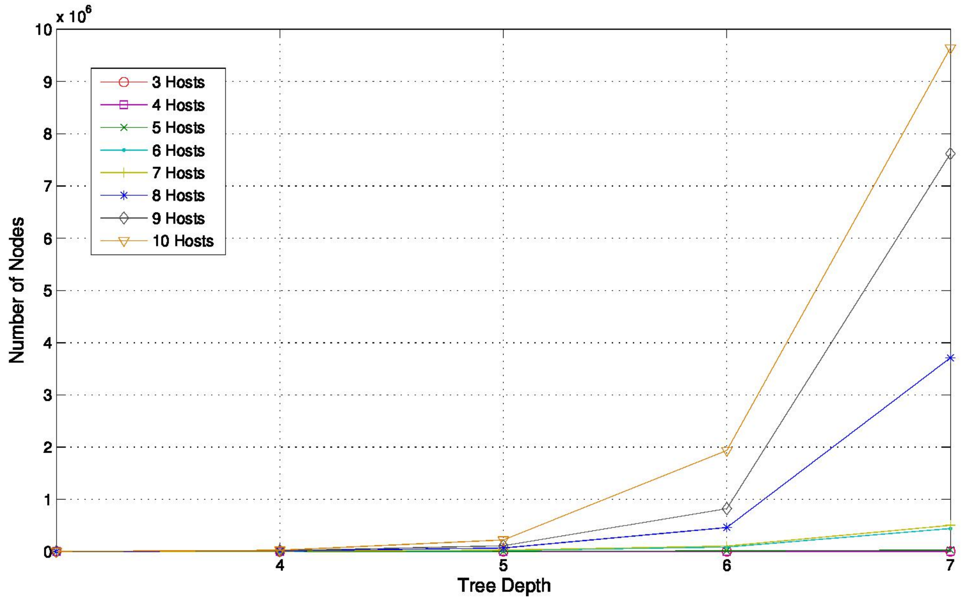
Given the interactive nature of this visualization and the relative low computational and spacial complexity at smaller depths for all graph sizes, we believe these algorithms should scale sufficiently and allow for a visualization at depths of at least five steps, sufficient to provide the user with valuable information that would be much more difficult to view in the case of a conventional attack graph. Once the visualization is re-centered, the tree can be recalculated given the position and previous steps. Figure 30 suggests that the tree generation time is well in order of rendering time for up to six steps.

5.4 Discussion and Limitations

This proposed visualization scheme could significantly improve the cyber-situational awareness capabilities of a network security administrator. Given the interactive nature of this model, we believe it can useful for many practical applications. It would strongly benefit situational perception (by mapping IDS alerts and displaying them on screen), impact assessment (by looking at future potential exploits the attacker may perform), situation



(a) Logarithmic Scale



(b) Linear Scale

Figure 29: Average Number of Nodes Relative to Tree Depth for Various Network Sizes

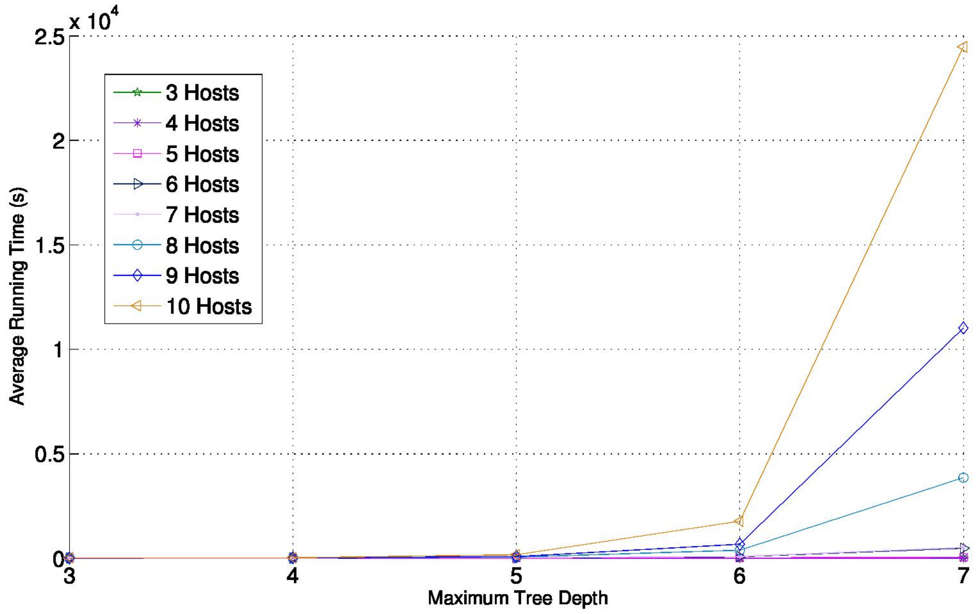


Figure 30: Average Computational Complexity in Function of Network Size and Tree Depth

tracking (live updating of the graph depending on an intruder's progress through the network), as well as plausible futures (by once again looking at the future potential exploits an attacker can use)- four of the seven of the aspects of situational awareness for cyber defense as defined by Barford et al.[BDD$^+$10].

Coupled with a real-time intrusion detection correlation system and mapping IDS alerts using the method described by Wang et al.[WLJ06]would allow the user to observe in real-time an attacker's intrusion progress through the network. At any given step, the visualization will display all possible next steps. While the possible explosion of states is a legitimate concern, capping the visualization at maximum depth ensures reasonable time and space complexity. As we have shown in our simulations, the visualization result and the running time are easily manageable with the maximal depth set to about six.

On the other hand, further study is needed to improve the tree expansion algorithms in order to avoid the exponential explosion, thus improving the practical maximum tree depth. Finding more efficient ways for incrementally updating the model after each centering operation, and improving the transition would improve the user experience.

Chapter 6

Conclusion

In this thesis, we have shown that information visualization is a crucial complement to the automated analysis of certain computer models, specifically attack graphs. Given the explosion of cloud services - in particular virtual private servers, where a hypervisor is the only separation between two different services - managing the complexity of attack graphs is of great importance.

Because existing visualization models are too broad through the adoption of a one-size-fits-all approach, and that heir scalability is limited, we believe that the introduction of application-specific visualization schemes for attack graphs will substantially improve the scalability as well as the readability and user comprehension of attack graphs. After discussing existing visualization techniques and presenting a hierarchical metric framework, we have introduced two new visualization paradigms, which should greatly assist a security analyst in obtaining a proper assessment of his network's security in two different and complementary use cases.

For the case of network overview, we have introduced a treemap-based connected ring visualization system: *radial attack treemaps*. By leveraging the scalability of displaying host configuration and vulnerabilities of treemaps and superior edge management of radial graphs, we have presented a novel, metric-driven, attack graph visualization scheme. We have demonstrated through implementation and large scale simulations that independently of subjective considerations, it offers superior scalability and information density.

In the case of situational awareness, we have introduced a topographic map-based tree visualization scheme: *topographic hyperbolic trees*. Hyperbolic trees allow for constant contextual awareness and high detail for the elements at the center of the visualization. Exhaustively listing all exploits in a tree can be a complex both in time and space, but our large scale simulation has shown us that this visualization can scale to up to five or six steps at a time.

Improving information visualization in all fields leads to an increase in the general understanding and usefulness of the model they illustrate. Attack graph is such a model, where exploits and conditions share sometimes complex relationships, and we believe these newly-introduced paradigms will substantially improve attack graph visualization, comprehension and usability for at least two use specific cases, further validating the versatility and effectiveness of the attack graph model.

While computers are essential to automated analysis, many actions and policies will very often be undertaken by a human analyst, and as such, his comprehension of a network's security posture is key to ensuring their networked systems and information's security. However powerful the model, it is only as good as what the user can extract from it.

Bibliography

[And07] James W Anderson. *Hyperbolic geometry*. springer, 2007.

[Art02] Michael Lyle Artz. *Netspa: A network security planning architecture*. PhD thesis, Massachusetts Institute of Technology, 2002.

[AWK02] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 217–224. ACM, 2002.

[BDD$^+$10] Paul Barford, Marc Dacier, Thomas G Dietterich, Matt Fredrikson, Jon Giffin, Sushil Jajodia, Somesh Jha, Jason Li, Peng Liu, Peng Ning, et al. Cyber sa: Situational awareness for cyber defense. In *Cyber Situational Awareness*, pages 3–13. Springer, 2010.

[Bel] Nicolas Garcia Belmonte. The javascript infoviz toolkit. `http://www.thejit.org`. Last Accessed: 02/03/2013.

[BETT94] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235 – 282, 1994.

[BG] Irad E. Ben-Gal. Bayesian networks. `http://www.eng.tau.ac.il/~bengal/BN.pdf`.

[BHW00] Mark Bruls, Kees Huizing, and Jarke J Van Wijk. *Squarified Treemaps*, volume pages, pages 33–42. Citeseer, 2000.

[Bou] Paul Bourke. Colour ramping for data visualization. `http://local.wasp.uwa.edu.au/~pbourke/texture_colour/colourramp/`. Last Accessed: 18/11/2012.

[BSW02] Benjamin B. Bederson, Ben Shneiderman, and Martin Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Trans. Graph.*, 21(4):833–854, October 2002.

[CIL$^+$10] Matthew Chu, Kyle Ingols, Richard Lippmann, Seth Webster, and Stephen Boyer. Visualizing attack graphs, reachability, and trust relationships with navigator. In *Proceedings of the Seventh International Symposium on Visualization for Cyber Security*, pages 22–33. ACM, 2010.

[CR74] Edwin Catmull and Raphael Rom. A class of local interpolating splines. *Computer aided geometric design*, 74:317–326, 1974.

[EGK$^+$01] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, Gordon Woodhull, Short Description, and Lucent Technologies. Graphviz - open source graph drawing tools. In *Lecture Notes in Computer Science*, pages 483–484. Springer-Verlag, 2001.

[FW08] Marcel Frigault and Lingyu Wang. Measuring network security using bayesian network-based attack graphs. In *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, pages 698–703. IEEE, 2008.

[FWSJ08] Marcel Frigault, Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring network security using dynamic bayesian network. In *QoP '08: Proceedings of the 4th ACM workshop on Quality of protection*, pages 23–30, New York, NY, USA, 2008. ACM.

[GS06] John Galloway and Simeon J. Simoff. Network data mining: methods and techniques for discovering deep linkage between attributes. In *In APCCM*

06: Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling, pages 21–32. Society, Inc, 2006.

[HEA] Hannes Holm, Mathias Ekstedt, and Dennis Andersson. Empirical analysis of system-level vulnerability metrics through actual attacks.

[HMM00] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *Visualization and Computer Graphics, IEEE Transactions on*, 6(1):24–43, 2000.

[HN09] Seok-Hee Hong and Hiroshi Nagamochi. New approximation to the one-sided radial crossing minimization. *Journal of Graph Algortihms and Applications*, 13(2):179–196, 2009. DOI: 10.7155/jgaa.00182.

[HN10] Seok-Hee Hong and Hiroshi Nagamochi. Approximation algorithms for minimizing edge crossings in radial drawings. *Algorithmica*, 58:478–497, 2010.

[Hol06] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12:741–748, 2006.

[HOS09] John Homer, Xinming Ou, and David Schmidt. A sound and practical approach to quantifying security risk in enterprise networks. *people. cis. ksu. edu*, 2009.

[HVOM08] John Homer, Ashok Varikuti, Xinming Ou, and Miles McQueen. Improving attack graph visualization through data reduction and attack grouping. *Visualization for Computer Security*, pages 68–79, 2008.

[IB12] Nwokedi Idika and Bharat Bhargava. Extending attack graph-based security metrics and aggregating their application. *Dependable and Secure Computing, IEEE Transactions on*, 9(1):75–85, 2012.

[Jaj10] Sushil Jajodia. *Cyber situational awareness: Issues and Research.* Springerverlag Us, 2010.

[Jaq07] Andrew Jaquith. *Security metrics: replacing fear, uncertainty, and doubt.* Addison-Wesley Professional, 2007.

[Jar73] Ray A Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18–21, 1973.

[JS91] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization '91, Proceedings., IEEE Conference on*, pages 284 –291, oct 1991.

[JSW02] Somesh Jha, Oleg Sheyner, and Jeannette M Wing. Minimization and reliability analyses of attack graphs. Technical report, DTIC Document, 2002.

[KP$^+$88] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.

[KSB$^+$09] Martin Krzywinski, Jacqueline Schein, İnanç Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J. Jones, and Marco A. Marra. Circos: An information aesthetic for comparative genomics. *Genome Research*, 19(9):1639–1645, September 2009.

[LAM$^+$05] Yarden Livnat, J. Agutter, S. Moon, R. F. Erbacher, and S. Foresti. A visualization paradigm for network intrusion detection. pages 92–99, 2005.

[LAMF05] Yarden Livnat, Jim Agutter, Shaun Moon, and Stefano Foresti. Visual correlation for situational awareness. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 95–102. IEEE, 2005.

[LRP95] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI

'95, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[LY06] Chun-Cheng Lin and Hsu-Chun Yen. On balloon drawings of rooted trees. In *Graph Drawing*, pages 285–296. Springer, 2006.

[LYL04] Kiran Lakkaraju, William Yurcik, and Adam J Lee. Nvisionip: netflow visualizations of system state for security situational awareness. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 65–72. ACM, 2004.

[MH98] G. Melancon and I. Herman. Circular drawings of rooted trees. In *Reports of the Centre for Mathematics and Computer Sciences*, 1998.

[Nis03] NIST Nist. 800-55, security metrics guide for information technology systems. *NIST paper*, 2003.

[NJ04] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 109–118. ACM, 2004.

[NJ05] Steven Noel and Sushil Jajodia. Understanding complex network attack graphs through clustered adjacency matrices. In *ACSAC*, pages 160–169, 2005.

[NRJ04] Steven Noel, Eric Robertson, and Sushil Jajodia. Correlating intrusion events and building attack scenarios through attack graph distances. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 350–359. IEEE, 2004.

[OBM06] Xinming Ou, Wayne F Boyer, and Miles A McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345. ACM, 2006.

[OFWB03] Joshua O'Madadhain, Danyel Fisher, Scott White, and Y Boey. The jung (java universal network/graph) framework. *University of California, Irvine, California*, 2003.

[OGA05] Xinming Ou, Sudhakar Govindavajhala, and Andrew W Appel. Mulval: A logic-based network security analyzer. In *14th USENIX Security Symposium*, pages 1–16, 2005.

[oST] National Institute of Standards and Technology. National vulnerability database. `http://nvd.nist.gov`.

[PBP02] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-spline techniques*. Springer, 2002.

[PS98] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM, 1998.

[RA00] Ronald W Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 156–165. IEEE, 2000.

[SA06] Ben Shneiderman and Aleks Aris. Network visualization by semantic substrates. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):733–740, 2006.

[Sch99] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[Sch05] Mike Schiffman. The common vulnerability scoring system (cvss), November 2005.

[She04] Oleg Mikhail Sheyner. *Scenario graphs and attack graphs*. PhD thesis, University of Wisconsin, 2004.

[SHJ+02] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002.

[Shn96] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. *Visual Languages, IEEE Symposium on*, page 336, 1996.

[SW01] B. Shneiderman and M. Wattenberg. Ordered treemap layouts. In *Information Visualization, 2001. INFOVIS 2001. IEEE Symposium on*, pages 73 –78, 2001.

[SW04] Oleg Sheyner and Jeannette Wing. Tools for generating and analyzing attack graphs. In *Formal methods for components and objects*, pages 344–371. Springer, 2004.

[TK91] William Thomson and Baron Kelvin. *Popular Lectures and Addresses. In Three Volumes.* 1891.

[War12] Colin Ware. *Information visualization: perception for design.* Morgan Kaufmann Pub, 2012.

[WF09] Leland Wilkinson and Michael Friendly. The history of the cluster heat map. *The American Statistician*, 63(2):179–184, 2009.

[WIL+08] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An attack graph-based probabilistic security metric. In *Proceeedings of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security*, pages 283–296, Berlin, Heidelberg, 2008. Springer-Verlag.

[WJSN10] Lingyu Wang, Sushil Jajodia, Anoop Singhal, and Steven Noel. k-zero day safety: Measuring the security risk of networks against unknown attacks. In *Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS'10)*, 2010.

[WLI08a] Leevar Williams, Richard Lippmann, and Kyle Ingols. Garnet: A graphical attack graph and reachability network evaluation tool. *Visualization for Computer Security*, pages 44–59, 2008.

[WLI08b] Leevar Williams, Richard Lippmann, and Kyle Ingols. An interactive attack graph cascade and reachability display. *VizSEC 2007*, pages 221–236, 2008.

[WLJ06] Lingyu Wang, Anyi Liu, and Sushil Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Comput. Commun.*, 29(15):2917–2933, 2006.

[WNJ06] Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, 2006.

[YYT$^+$04] Xiaoxin Yin, William Yurcik, Michael Treaster, Yifan Li, and Kiran Lakkaraju. Visflowconnect: netflow visualizations of link relationships for security situational awareness. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 26–34. ACM, 2004.