Mitigating Yo-Yo Attacks on Cloud Using Game-Theoretical Modelling and Learning-Based Approach

Saman Saniee Monfared

A Thesis

in

The Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Quality Systems Engineering) at

Concordia University

Montréal, Québec, Canada

December 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Saman Saniee Monfared**

Entitled: **Mitigating Yo-Yo Attacks on Cloud Using Game-Theoretical Modelling and Learning-Based Approach**

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

Master of Applied Science (Quality Systems Engineering)

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

Signed by the Final Examining Committee:

______________________________ Chair
Dr. Rachida Dssouli

______________________________ Examiner
Dr. Farnoosh Naderkhani

______________________________ Supervisor
Dr. Jamal Bentahar

Approved by ______________________________
Dr. Jun Yan, Chair
Department of The Concordia Institute for Information Systems Engineering

____________ 2023

____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Mitigating Yo-Yo Attacks on Cloud Using Game-Theoretical Modelling and Learning-Based Approach

Saman Saniee Monfared

Cloud computing, a transformative paradigm, has ushered in an era of unparalleled convenience and economic efficiency for both service providers and users. Historically, before its widespread adoption, digital services were susceptible to Distributed Denial of Service (DDoS) attacks, which aimed to overwhelm server capacities. The advent of cloud computing has primarily mitigated these threats but, in doing so, has inadvertently introduced new vulnerabilities. In their relentless pursuit of exploitation, attackers have transitioned from targeting server performance to inflicting economic damages. A particularly insidious form of this is the Yo-Yo Attack, an Economic Denial of Sustainability (EDoS) strategy that manipulates the auto-scaling features inherent to cloud systems.

This thesis presents a novel mathematical framework to understand the ongoing tussle between cloud service providers and Yo-Yo Attackers. We conceptualize this conflict as a Repeated Dynamic Bayesian Stackelberg game. This approach is pioneering in capturing the Yo-Yo attacker's nuanced strategies, particularly their adeptness at exploiting the cloud's auto-scaling features. The Learning-Based Attackers' Type Recognition and Defense Mechanism is central to our game model, which harnesses the power of one-class Support Vector Machine (SVM) to discern the modus operandi of various Yo-Yo attack types.

Our research further introduces an innovative machine-learning algorithm adept at recognizing and countering the unique attack patterns of individual bots. We delve deeper into a proposed defense mechanism, which recognizes different strategies of Yo-Yo attackers aiming to exploit different vulnerabilities of the cloud's auto-scaling mechanism and thereby confound their efforts. Our empirical experiments underscore the efficacy of our solution. Our approach demonstrates superior reduced compromised services and overall efficiency compared to existing Yo-Yo detection and defense strategies.

In conclusion, as cloud computing continues to dominate the digital landscape, ensuring its security remains paramount. Our research offers a robust solution to a new generation of threats, setting a benchmark for future endeavors in cloud security.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to Prof. Jamal Bentahar for his unwavering guidance, support, and mentorship throughout the course of this research. His expertise, patience, and encouragement have been invaluable to me, especially during the unprecedented challenges posed by the COVID-19 pandemic. I am truly fortunate to have had him as my supervisor during these trying times.

The journey of completing this thesis under the constraints of quarantine and the shift to almost entirely remote learning has been a unique and challenging experience. I am immensely grateful for the adaptability and resilience shown by the academic community and for the innovative tools and methods that have been developed to facilitate our learning and research.

I would also like to extend my heartfelt thanks to my family, who have provided me with endless love, understanding, and encouragement throughout my academic journey, particularly during the isolation of the pandemic. Their belief in my capabilities and their unwavering support have been the pillars upon which I have built my aspirations.

To my friends, thank you for being my sounding board, for the countless brainstorming sessions, and for always being there to share in both the challenges and the triumphs. Your camaraderie and support, even from a distance, have made this journey all the more memorable.

Lastly, I am grateful to all those who have indirectly contributed to this work and have been a part of my academic journey. Your influence, in ways big or small, has shaped this thesis and my experiences during its creation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 About Cloud

In recent years, cloud computing has emerged as a transformative force in the field of information technology, rapidly expanding its reach and influence across various sectors. This paradigm shift is primarily driven by the multitude of benefits it offers, such as cost efficiency, enhanced scalability, and improved resource management. Moreover, the unprecedented challenges posed by the COVID-19 pandemic have further hastened the transition from in-house infrastructure to cloud-based solutions [1] as businesses and individuals alike sought to adapt to the new normal of remote work and virtual collaboration. The transition of companies from investing in their own in-house infrastructures to outsourcing their computation needs to third-party cloud providers has had numerous advantages for both sides [2]. This shift is motivated by the inherent benefits of cloud computing and the challenges and costs associated with managing and maintaining in-house systems.

a. Cloud Advantage

To attain a comparable level of Quality of Service (QoS) without relying on cloud services, businesses, and online companies would have to invest substantially in infrastructure, maintenance, and skilled personnel. Building and maintaining an in-house data center with a high QoS involves significant upfront costs for hardware acquisition, space requirements, and power consumption. Additionally, organizations must allocate resources to employ expert staff responsible for ongoing infrastructure maintenance, security, and management. This includes addressing the inevitable need for replacing failed hard disk drives and processors, which adds to the overall burden of managing a data center.

Security is another crucial concern for businesses managing their own infrastructure. Ensuring a robust security posture requires continuous monitoring, threat detection, and mitigation efforts. This demands a dedicated team of security professionals who are well-versed in the latest threat landscape and capable of implementing comprehensive defense strategies. The rapid advancements in machine learning and AI, the proliferation of cyberattacks, and the fact that attackers constantly adapt to security measures are undeniable challenges businesses face in securing their infrastructure.

There are several occasions when each online business might experience the urge to make temporary or permanent adjustments to its infrastructure to serve its users better. For instance, suppose a business intends to launch new services or test new features for future developments. In that case, it usually requires additional hardware upgrading to increase the capacity of its servers or timely software updates, which incurs additional costs. Furthermore, there are certain times of the year when online businesses experience spikes (fluctuations) in user demand, such as holiday sales for e-commerce businesses or major content releases on streaming services. Before the advent of cloud computing's auto-scaling feature, it was nearly impossible for businesses to prevent crashes or service slowdowns during such spikes without investing heavily in additional servers, which would then sit idle during periods of average user traffic.

Regarding data safety, an in-house solution would require implementing multi-regional backups and disaster recovery plans to safeguard against catastrophes, power outages, and other unforeseen events. This necessitates a complex and costly infrastructure that spans multiple geographical locations and dedicated personnel to manage the backup and recovery process. The challenge lies in synchronizing data across all locations, ensuring timely recovery, and maintaining the integrity of the data during the process.

Accessibility and data synchronization are also vital considerations for companies with multiple branches and offices. Ensuring seamless, real-time access to data across various locations would entail significant investment in networking, infrastructure, and data management solutions. The need for instantaneous data synchronization across multiple locations introduces additional complexities and potential bottlenecks that may impact overall performance and user experience. Given these challenges, cloud computing emerges as an attractive alternative for businesses seeking to achieve the above benefits without incurring the substantial costs and complexities associated with in-house infrastructure. The integration of cloud services in online businesses has proven to be highly advantageous due to its inherent cost-efficiency and scalability. By adopting a pay-as-you-go model, customers and providers both experience substantial cost savings, while auto-scaling features enable seamless accommodation of fluctuating user demands. Additionally, cloud services facilitate instant updates and upgrades, allowing for accelerated testing, launching of new products, and rapid feature implementation.

Furthermore, cloud computing offers enhanced security measures, as cloud providers invest heavily in cutting-edge security technologies and employ dedicated teams of experts to protect their client's data. This level of security is often difficult for individual businesses to achieve with their in-house infrastructure. Moreover, cloud services provide robust data safety protocols, including multi-regional backups and disaster recovery plans, which alleviate the challenges associated with safeguarding data in the face of unforeseen events.

Lastly, the cloud enables seamless accessibility and data synchronization across multiple locations, empowering businesses to embrace a more agile and flexible operational model. This is particularly beneficial for companies with multiple branches and offices, as it ensures real-time access to data and eliminates potential bottlenecks that may arise due to complex data management and synchronization requirements.

b. Cloud Auto-Scaling Vulnerabilities

Every technological advancement comes with its share of risks, and cloud computing is no exception. One of the vulnerabilities associated with cloud computing lies in its auto-scaling feature [3]. While auto-scaling provides numerous benefits, it also exposes cloud systems to new types of attacks.

The vulnerability of the auto-scaling feature can be exploited by adversaries, potentially leading to detrimental effects. A specific instance of such exploitation is embodied as the ’Yo-Yo’ attack, a form of Distributed Denial-of-Service (DDoS) attack that manipulates the auto-scaling mechanism of cloud computing systems.

1.1.2 Yo-Yo Attack

Distributed Denial of Service (DDoS) attacks have long threatened online businesses, overwhelming targeted systems with a flood of traffic and rendering them inaccessible to legitimate users [4, 5]. Cloud computing can mitigate the impact of DDoS attacks due to its auto-scaling mechanism [6], which can automatically scale resources to accommodate increased traffic. However, this same feature can also be exploited by malicious actors differently [7, 8].

Economic Denial of Sustainability (EDoS) attacks are a specific type of attack that aims to exploit the auto-scaling vulnerability in cloud systems [9–11]. These attacks focus on causing financial damage to the targeted organization by continuously generating fake traffic, forcing the auto-scaling mechanism to scale resources and, in turn, increasing operational costs.

Yo-Yo attacks are a subtype of EDoS attacks that involve alternating between high and low traffic periods, making it more difficult for cloud providers to distinguish between legitimate and malicious traffic. This deceptive attack pattern can cause businesses significant performance and financial damage by continuously triggering the auto-scaling mechanism and increasing infrastructure costs.

1.1.3 Game Theory

Game Theory is a branch of mathematics that studies decision-making in situations where multiple agents interact with each other, taking into account the strategic behavior of each agent. It has been widely used to model and analyze complex interactions in various fields, including economics, political science, and cybersecurity.

a. Game Theory Overview

Game theory is a mathematical approach to modeling and analyzing situations in which multiple decision-makers, or players, interact with each other strategically. It seeks to understand and predict players' choices under various conditions, considering the incentives, preferences, and information available to them. Game theory applies to a wide range of fields, including economics, political science, biology, and computer science, and is especially useful in analyzing situations where decision-makers have conflicting interests.

The fundamental concept in game theory is that of a game. A game is typically defined by its players, strategies, and payoffs. Players are the decision-makers involved in the game, each having their own objectives and preferences. Strategies are the set of actions or choices available to the players. Payoffs represent the consequences or rewards associated with the combinations of strategies chosen by the players. A key objective of game theory is to identify the best strategies for each player, given their preferences and the strategies available to other players in the game.

b. Nash Equilibrium

An important concept in game theory is that of a Nash equilibrium. A Nash equilibrium is a situation in which no player can improve their payoff by unilaterally changing their strategy, given the strategies chosen by the other players. In other words, each player's strategy is optimal, considering the other players' strategies. Nash equilibrium can be found in both simultaneous and sequential games and serves as a standard solution concept in game theory.

c. Bayesian Games

There are two main types of games in game theory: simultaneous and sequential. In simultaneous games, all players make their decisions simultaneously without knowing the other player's choices. In sequential games, players make decisions in a particular order, with each player's decision potentially affecting the choices of the subsequent players. Sequential games are often represented using game trees, which depict the order of play, available strategies, and payoffs at each stage.

Bayesian games are a subclass of games that incorporate incomplete information. In a Bayesian game, players have private information about their types, which can represent their preferences, abilities, or other relevant characteristics. Players form beliefs about the types of other players based on a probability distribution, which influences their strategic choices. The goal of Bayesian game theory is to find the optimal strategies for each player, given their type and beliefs about the types of other players.

d. Repeated Games

In a repeated game, players interact with each other multiple times, making decisions in a series of discrete periods or stages. Repeated games are particularly relevant when players have the opportunity to learn from their past interactions and adjust their strategies accordingly. One common solution concept for repeated games is the subgame perfect Nash equilibrium (SPNE), a strategy profile that constitutes a Nash equilibrium in every subgame, including the overall game itself. This solution concept helps identify credible strategies that account for the dynamic nature of repeated games.

Discount factors are used in repeated games to account for the time value of payoffs. A discount factor represents the relative weight a player assigns to future payoffs compared to immediate ones. In a repeated game with an exponential discount factor, each player's utility function incorporates the discounted sum of their payoffs over all the game periods. By adjusting the discount factor, players can more or less emphasize immediate rewards relative to future ones.

e. Bayesian Stackelberg Games

Stackelberg games are a specific type of sequential game in which one player, called the leader, moves first, and the other player called the follower, moves second. The leader commits to a strategy, and the follower observes it and chooses their best response accordingly. The Stackelberg equilibrium is a solution concept that identifies the optimal strategies for both the leader and the follower, given their preferences and the sequential structure of the game.

In a Bayesian Stackelberg game, the leader and the follower have incomplete information about each other's types. The leader forms beliefs about the follower's type based on a probability distribution and chooses a strategy that maximizes their expected utility, considering the follower's best response function. The follower, in turn, observes the leader's strategy and chooses their best response based on their type and beliefs about the leader's type. The Bayesian Stackelberg equilibrium is a strategy profile that specifies the optimal strategies for both the leader and the follower, given their types, preferences, and beliefs about each other's types.

Mixed strategies are another important concept in game theory. A mixed strategy is a probability distribution over a player's set of pure strategies, indicating the likelihood of choosing each strategy. In some games, it can be advantageous for players to employ mixed strategies rather than committing to a single pure strategy. Mixed strategies can introduce uncertainty and unpredictability into the game, making it more difficult for opponents to exploit weaknesses in a player's strategy.

f. Applying Game Theory to the Problem

A repeated Bayesian Stackelberg game can be formulated to apply game theory to the problem of cloud provider-attacker interaction in the context of Yo-Yo attacks. The cloud provider acts as the leader, choosing defense strategies based on their beliefs about the attacker's type. In contrast, the attackers act as the followers, choosing attack strategies in response to the cloud provider's defense strategies. Both players have utility functions incorporating exponential discount factors, emphasizing the importance of recent rewards over older ones.

In this game, the cloud provider's and attacker's utility functions are used to formulate maximization problems for each player, subject to constraints on their policies (defense and attack strategies) and other relevant factors. The Bayesian Stackelberg game is then solved using backward induction, combining the attacker's maximization problem with the cloud provider's maximization problem. This results in a Mixed-Integer Quadratic Programming (MIQP) problem, which is then converted into a Mixed-Integer Linear Programming (MILP) problem to find the optimal defense strategy for the cloud provider.

By learning and understanding these game theory concepts and their application to the problem at hand, one can gain a deeper insight into the strategic interaction between cloud providers and attackers in the context of Yo-Yo attacks. This knowledge can ultimately contribute to developing more effective defense strategies and countermeasures against such attacks, ensuring the continued security and reliability of cloud services for all users.

1.2 Problem Statement

To further illustrate the application of game theory to the problem of mitigating Yo-Yo attacks by the cloud, let us consider a simplified example. Suppose there are two types of attackers, with different attacking capabilities, and the cloud provider has two defense strategies. The cloud provider estimates the probability distribution of facing each attacker type and uses this information to determine the best defense strategy. On the other hand, the attacker observes the defense strategy chosen by the cloud provider and selects their attack strategy accordingly.

In this example, the cloud provider would first solve their maximization problem, considering the expected utility resulting from each possible combination of defense and attack strategies. Then, the attacker's maximization problem would be integrated into the cloud provider's problem through backward induction. This would ultimately result in an MIQP problem, which could be converted to a MILP problem and solved to identify the optimal defense strategy for the cloud provider.

This example is, of course, a simplification of the actual problem addressed in the thesis. However, it illustrates the general approach and methodology used in applying game theory to the study of Yo-Yo attacks. By developing a comprehensive understanding of the underlying game theory concepts and their relevance to the problem at hand, researchers and practitioners can make more informed decisions regarding developing and implementing effective defense strategies against such attacks.

Having now covered the basics of game theory and its application to the specific problem of Yo-Yo attacks, it is essential to understand the significance and implications of the proposed Stackelberg Bayesian game model. This model allows for a more thorough understanding of the strategic interactions between cloud providers and attackers, ultimately leading to the development of more effective defense strategies.

One of the key benefits of employing a game theoretic approach is that it provides a systematic and mathematically rigorous framework for analyzing the decision-making processes of both attackers and defenders. By identifying the optimal strategies for both parties under various conditions, game theory enables researchers to understand the factors influencing their behavior and the consequences of their actions.

Furthermore, by modeling the problem as a Stackelberg Bayesian game, the proposed approach accounts for the inherent asymmetry in the roles of the cloud provider (leader) and attacker (follower). This is important because it reflects the real-world dynamics of the problem, where the cloud provider typically has more information about the system and makes decisions that influence the attacker's subsequent actions.

In conclusion, the application of game theory to the study of Yo-Yo attacks and cloud security, in general, provides valuable insights that can contribute to the development of more effective and resilient defense strategies. By thoroughly understanding the underlying concepts and methodologies involved in game theory, researchers and practitioners can make better-informed decisions and more effectively safeguard the security and reliability of cloud services.

To summarize, game theory offers a valuable framework for understanding and analyzing the strategic interactions between cloud providers and attackers in the context of Yo-Yo attacks. The Stackelberg Bayesian game model, in particular, allows us to consider the asymmetric information and roles of the two players, ultimately leading to more effective defense strategies. The proposed model incorporates critical elements such as mixed strategies, pure strategies, probability distributions, and optimization problems, making it a comprehensive and powerful tool for studying this complex problem.

With this comprehensive overview of game theory, we now have a solid foundation to understand the Stackelberg Bayesian game model used in this thesis and how it can be applied to analyze the strategic interactions between cloud providers and attackers in the context of Yo-Yo attacks.

1.3 Contributions

The main contribution of this thesis is modeling the Yo-Yo attack as a Repeated Dynamic Stackelberg Bayesian game. Then, we employ this model to optimize the cloud defense strategy against Yo-Yo attackers to mitigate the damages of their attacks better.

In summary, the main contributions of this work are:

- Characterizing the nature of Yo-Yo attacks and formulating these attributes into a mathematical model. This first step towards systematizing the defense against such attacks provides a critical foundation for the rest of the study.
- Introducing proper variables that correspond to the motivations and objectives of the Yo-Yo attackers. These variables allow for a more nuanced understanding of the attack dynamics and better equip the cloud provider in its defense.
- Defining different types of attacking strategy prioritization for the Yo-Yo attackers enables them to exploit the Auto-Scaling vulnerabilities in a more sophisticated and more intelligent manner.
- Designing a Repeated Dynamic Stackelberg Bayesian game that eventually leads the cloud provider to implement the optimal defense strategy against Yo-Yo attacks.
- Solving the game model using MILP to find the best strategy for the cloud provider. This approach enhances computational efficiency and optimality, supporting a more robust defense against Yo-Yo attacks.
- Proposing a framework to categorize different types of Yo-Yo attackers to increase the accuracy and efficiency of cloud security against such attacks. By recognizing the differences among attackers, the cloud provider can tailor its strategy according to the specific type of Yo-Yo attack it faces.

This novel approach will provide a deeper understanding of the interactions between attackers and cloud providers in the context of this specific type of EDoS attack, ultimately paving the way for developing more effective strategies to mitigate and prevent such periodic attacks in the future.

Chapter 2

Related Work

2.1 Cloud Auto-Scaling & Vulnerabilities

The authors In [12] delved into the vulnerabilities of cloud auto-scaling systems, particularly emphasizing the ”Yo-Yo attack”. While the cloud’s auto-scaling feature is adept at handling traditional Distributed Denial of Service (DDoS) attacks [13], it inadvertently introduces susceptibility to Economic Denial of Sustainability attacks (EDoS) [14] and Reduction of Quality (RoQ) attacks [15]. The ”Yo-Yo attack,” a term first coined in this research, is an efficient assault on the auto-scaling mechanism. It operates by cycling between two phases: the on-attack phase, where a short burst [16] of traffic prompts the auto-scaling to scale up, and the off-attack phase, initiated when the attacker ceases the excess traffic upon recognizing the scale-up. This cyclical process can keep an adaptive mechanism in a perpetual oscillation state between over-load and under-load conditions. The researchers utilized Amazon’s cloud service for their experimental evaluations. Their model revealed that the attacker spent approximately 77% of the time in the off-attack phase, leaving the victim in constant flux. The findings indicate that the Yo-Yo attack is thrice as potent as a consistent DDoS attack, underscoring its potential threat in cloud environments. The paper underscores the need for a nuanced understanding of these policies, especially the differences between adaptive and discrete, to safeguard against such sophisticated attacks. The authors advocate for early scale-up and slower scale-down configurations as a countermeasure.

Building on the foundational work of [12], The author in [17] further elucidated the intricacies of the Yo-Yo attack on cloud auto-scaling mechanisms. Their experimental evaluations of Amazon's cloud service highlighted the differential impacts of two auto-scaling policies: adaptive and discrete. Their findings revealed that while the adaptive policy responded quicker, it suffered from more significant economic and performance damages compared to the discrete policy. This was attributed to the substantial warm-up time when users incurred costs for all the machines without any performance enhancement. Furthermore, their analysis showcased the Yo-Yo attack's economic efficiency, as it could inflict more damage per unit cost than a traditional DDoS attack. This economic advantage makes the Yo-Yo attack appealing to potential attackers, especially considering the real-world implications of renting botnets for such assaults. The study's experimental results, especially those related to the adaptive policy, indicated that the Yo-Yo attack could deploy twice the number of machines at half the cost of a DDoS attack, emphasizing the critical need for robust defense strategies against such sophisticated threats [15, 18]. Building on the previous insights provided in [12, 17], the authors in [19] present an innovative approach to mitigate the Yo-Yo attack in cloud auto-scaling mechanisms. They introduce the TASD system, which leverages the trust values of users to identify adversarial users. Utilizing the CloudSim toolkit for their experimental evaluations, the results indicate that TASD can detect up to 80% of malicious requests with a false positive rate of only 6.7%. Furthermore, TASD effectively disrupts the cyclical nature of the attacker's response time, making it challenging for attackers to infer the state of the auto-scaling mechanism through probe packets. The system's efficiency is further underscored by its ability to reduce the number of attacks by 38% and decrease scale-ups by 41%. However, the authors also highlight a trade-off: while higher trust value thresholds can reduce the number of attacks, they can also lead to a higher false positive rate. This paper's findings underscore the potential of trust-based systems like TASD in defending against sophisticated Yo-Yo attacks, complementing the earlier works by offering a tangible solution to a previously identified vulnerability.

Expanding upon the foundational understanding of Yo-Yo attacks, the study presented in [20] offers a comprehensive exploration of the Trust-based Adversarial Scanner Delaying (TASD) system in a natural cloud production environment, specifically Amazon Web Services (AWS). The original TASD system, as introduced in [21], utilized an additive decrease method to update trust values dynamically. Drawing inspiration from TCP rate control mechanisms, the authors introduced two optimized methods: ADAI (Additive Decrease/Additive Increase) and MDAI (Multiplicative Decrease/Additive Increase). These methods aim to enhance the TASD system's detection and mitigation capabilities. The paper demonstrates through a series of experiments that the proposed TASD variants can effectively identify and counteract Yo-Yo attacks in real cloud applications. The results emphasize the system's ability to stabilize instance scaling after detecting an attack, ensuring consistent service delivery and minimizing the oscillatory effects of the Yo-Yo attack. However, it is noteworthy that this work primarily focuses on a singular type of exploitation of the cloud's auto-scaling vulnerability and does not encompass various Yo-Yo attacking strategies, signifying the need for more comprehensive solutions, such as those presented in our research.

2.2 Game-Theoretic Approaches for Cloud's Security Applications

Building upon the foundational studies on the vulnerabilities of cloud auto-scaling systems, in [22] the writer introduced a sophisticated detection and defense mechanism tailored for cloud-based systems. Their approach is anchored in a game-theoretical model, specifically a repeated Bayesian Stackelberg game. This model is designed to address multi-type attacks in the cloud. The authors proposed a multi-phased strategy: (1) a risk assessment framework to evaluate the risk level of each guest Virtual Machine (VM); (2) a Moving Target Defense (MTD) mechanism that intelligently migrates services from vulnerable VMs to more secure ones; (3) a machine learning technique to identify attacker types using honeypot data; and (4) a resource-aware Bayesian Stackelberg game to aid the hypervisor in determining the optimal detection load distribution strategy among VMs [23]. Their experiments, conducted using Amazon's data center, AWS honeypot [24] data, and the CloudSim toolkit [25], demonstrated the efficacy of their solution in improving detection performance and minimizing attacked services. However, a critical observation is that while their system might be adept at detecting and diverting potential threats to honeypots, it remains susceptible to some types of Economic Denial of Sustainability (EDoS) attacks, especially the Yo-Yo attacks. The inherent challenge is that even if the Yo-Yo attackers are redirected to the honeypots, the cloud provider still incurs financial costs. The Yo-Yo attackers, not necessarily aiming to access crucial data, can still financially strain the cloud system. Running honeypots, while a deterrent, still requires financial resources for its infrastructure [26].

Given these observations, our work emerges as a more comprehensive solution against such EDoS attacks. While [22] provides a robust foundation for detecting and diverting threats, our approach is designed to detect and effectively counteract the financial implications of sophisticated attacks like the Yo-Yo attack. Our model offers a more holistic defense mechanism, ensuring cloud systems' security and economic sustainability.

In [27], an optimal defense allocation strategy is presented, considering interdependencies, such as assets under a single vendor. These interdependencies are represented through an interdependency graph. The model assumes attackers exploit these interdependencies to target high-value assets. The game involves multiple defenders, each tasked with safeguarding a group of assets. Each defender aims to reduce its potential losses, considering that attack probabilities on its assets are influenced by its defense measures, other defenders' actions, and the interdependency graph.

In [28], a game-theoretical framework is developed between an external threat and a network of decoy nodes, structured in two phases. The initial phase examines the interactions between the adversary and an individual decoy node. This phase focuses on the adversary's attempts to (1) discern the decoy node by analyzing node response timings and (2) detect discrepancies in protocol implementations to distinguish between real and decoy nodes. The result of this phase determines the duration an adversary needs to ascertain if a node is genuine or a decoy. The subsequent phase formulates games where the adversary aims to identify genuine nodes within a mixed network of real and decoy nodes. This phase's outcome suggests the system's best strategy to shuffle the IP address space, making it challenging for the adversary to pinpoint actual nodes.

Chapter 3

Game Theoretic Problem Modelling Formulation

3.1 Model Components

3.1.1 Model Overview

Two significant roles are on each side of this game: the cloud provider and the cloud's users. On one side is the cloud provider, a business entity aiming to deliver infrastructure services for profit. This player seeks to maximize efficiency, finding the delicate balance between cost-saving enforcement and the necessity of maintaining user satisfaction. The ultimate goal of the cloud provider is to accumulate benefits over time, building a sustainable and profitable business model.

On the other side are the users, who seek reliable services with minimal latency and errors. They are the final consumers of the cloud provider's services, and their satisfaction is paramount to its success. Between the cloud provider and the final users, there might be middlemen, also known as the cloud's customers. They are the service providers that utilize cloud infrastructure to present their services to the end users. The security measures in the interaction between the cloud and the attacker are not directly related to the service running on the cloud servers. However, instead, they relate to the auto-scaling criteria set by these middleman services.

Finally, a potential malicious group exists among the users, acting as a major periodic Distributed Denial-of-Service (DDoS) attack, also known as the Yo-Yo attack or a Reduction of Quality (RoQ) attack. An analogy could be drawn here between the Yo-Yo attack and a tumor in a living organism. The cloud provider acts as the body, and the attacker bots (the individual elements of the Yo-Yo attack) are like tumor cells feeding off the body. These bots exploit the cloud's resources and disrupt its operations, much like how tumor cells invade and impair bodily functions.

With these elements in place, the stage is set for a dynamic game of strategies and counter-strategies, with the cloud provider continually seeking ways to protect its infrastructure and ensure service quality. The attackers perpetually search for vulnerabilities to exploit. This game's end goal for the cloud provider is to maintain its service integrity while maximizing user satisfaction and profitability.

3.1.2 The Cloud

The first player is the cloud provider, which we refer to as the cloud. The cloud can monitor specific activity metrics from its active users to ensure any malicious Yo-Yo attacker is not exploiting the servers. By saying cloud, we assume it is an entity that can analyze the users' activity metrics such as CPU utilization, in/out network traffic in bytes, disk read/write in bytes or operations, etc., and decide if blocking specific user bots due to their repeated malicious activities, especially during the critical times helps to enhance the overall security of the servers.

Decision-making in the cloud can be done by its brain, load balancer, VM manager, etc. We mean the 'Cloud' as one entity with every ability and authority necessary to make decisions and act on its scaling strategies, monitor and analyze users' activities metrics, block malicious users, implement defense strategy mechanisms, move data among VMs, etc.

We observe this game through the eyes of a cloud provider who tries to pick an optimum defense strategy that idealistically can help it gain maximum profit by providing the highest and fastest quality of service to the normal users, which results in growth in their satisfaction ratings, the number of customers, and their daily average usage time and overall more accumulated benefit from users for them. At the same time, the cloud wants to minimize the costs resulting from unnecessary scale-ups caused by malicious Yo-Yo attacker bot activities. Over time, it wants to block the users that it finds malicious, acting as the Yo-Yo attacking bots. However, the defense and blocking against the Yo-Yo attackers also have associated costs that the cloud seeks to minimize as much as possible to have maximum profit while maintaining the safety of the servers against Yo-Yo attacks.

The cloud aims to isolate/block the malicious users that act as Yo-Yo attacker bots from mingling with normal users to help maintain providing a consistent, safe service to the legit normal users and minimize attack damage costs.

The goal of the cloud provider in our model is to maximize its profit by obtaining the financial benefit that each user delivers to the cloud by maximizing the detection and blocking of these malicious user bots while maintaining maximum accuracy and minimum error. The success in achieving this goal results in minimizing unnecessary Scaling costs and subsequently, maximizing the profit (payoff) of the cloud.

3.1.3 The Users

On the other side of the game, there are cloud users who want to use a desirable cloud service [29] with minimum response latency and connection problems from the servers together at a reasonable price. The cloud provides its users with services to obtain their financial benefit for its profit. This benefit can be derived from different sources; it can come from advertising, subscriptions, etc., or it can come directly from the service provider's funds. In any way, the aim is to show that to make the game model work, there must be a benefit incentive for the cloud to be provided to the users.

This benefit is a function of users' value for the cloud provider and is not necessarily equal for all users. For services that work with a pay-as-you-go model or show ads to users, the more a user uses the services, the more benefit goes to the cloud. Other factors like geographical location can also determine the benefit amount from each user. For, displaying ads to YouTube users in a higher GDP country is likely to be more costly than in those low GDP countries. Overall, the benefit of a user for the cloud is directly related to the amount of money that the user can eventually generate for the cloud provider by using the running service on those cloud servers. At the end of the day, the cloud wants to make the maximum profit, which means that the higher the users accumulated benefits than their costs, the better for the cloud.

Between the cloud provider and the final user, other joints might also be involved as middlemen. By middleman, we mean any online service, e.g., websites, phone applications, software, etc., that uses cloud infrastructure to host services to the end users. However, the interaction between the attacker and the cloud is unrelated to the service running on the cloud servers. Still, It is directly related to cloud auto-scaling criteria. The Yo-Yo attack and, in general, the EDoS attacks are focused on the auto-scaling feature vulnerability related to the cloud management algorithms. These are either defaulted by the cloud's underlying managing algorithms or can be adjusted within the allowed corresponding threshold range during the overload and underload periods by the cloud customer (middleman). In a nutshell, we mean the cloud customer as the service that implements cloud infrastructure to present its service to the end users.

3.1.4 The Attackers within The Users

There is always a chance that a group within the users does suspicious or malicious coordinated activities. Unlike conventional DDoS attacks in the cloud, where the impact can often be mitigated by the considerable cost incurred by the attacker, the Yo-Yo attack offers a more cost-effective disruption method, making it more challenging to track. The essence of a Yo-Yo attacker's strategy is exploiting the auto-scaling mechanism's responsive nature and employing a cyclical pattern of activities designed to force the cloud system into futile cycles of server overload, scale-up, and scale-down.

The Yo-Yo attack operates in two alternating phases. In the 'On-attack' phase, the attacker initiates a short burst of traffic through its bots significant enough to trigger the overload period necessary for the auto-scaling mechanism, leading to a scale-up. This causes the cloud system to allocate extra resources to handle the surge, resources that could be more beneficial as they are engaged in processing bogus traffic. In the 'Off-attack' phase, the attacker ceases sending excess traffic, waiting for the system to identify the diminished demand and initiate a scale-down. And repeat this process over and over.

The potency of the Yo-Yo attack lies in its iterative nature. As soon as the attacker identifies a scale-down, they repeat the process, thus maintaining a constant strain on the system. The ability of the attacker to discern the appropriate timing to switch between these two phases significantly influences the attack's strength. By mastering this timing, an attacker could maximize the disruption to the system while minimizing their own cost. In our model, we consider that the attacker can access the scaling state of the cloud servers.

The cloud may detect its scaling state by sending probe packets to the cloud's servers and analyzing its response time. A longer than usual response time could hint that the servers are under overload and are in the period right before a scale-up. A shorter-than-usual response time can address the attacker that the cloud is already scaled up. However, because there is no real need for that excess computation capacity, the cloud is before an expected scale-down is triggered. Moreover, the default scaling criteria of the cloud providers, which can be visible to every cloud customer, can help the attackers figure out the exact scaling state of the cloud.

This manipulation pattern of the Yo-Yo attack essentially exploits the cloud system into paying for unproductive resources, creating an avenue for financial drain alongside performance disruption for the cloud's customers (middlemen). It exemplifies how system features intended for efficiency can, under malicious intent, be reoriented into vulnerabilities. Thus, it is of paramount importance for the cloud to incorporate robust safeguards against such exploitative attacks within the auto-scaling mechanism.

Suppose a cloud provider business bleeds much cash due to the financial damage of paying for unnecessary extra server scale-ups. In that case, it results in increasing the overall price of the cloud for their customers. And thus decreasing the satisfaction of the general customers in the end. That explains the motivation of the cloud provider to block the users with malicious behaviors that contribute to a Yo-Yo attack.

3.1.5 Attacking Brains

As mentioned above, each Yo-Yo attack consists of periodic particles of DDoS attacks, which inherently means that the Yo-Yo attack is done through multiple malicious bots oriented to sabotage the host's servers, similar to DDoS. However, this time, it uses DDoS to force Yo-Yo attacks with the maximum attack potency, maximizing the damage while minimizing the costs. So, although there might be many users (either malicious bots or manipulated exploited machines from reasons such as IoT Internet of Things) contributing to a Yo-Yo attack, the cloud wants to detect the significant strategies that lead those attacker bots.

This herd behavior nature of the Yo-Yo attacker bots leads us to understand the fact that in order to detect and stop Yo-Yo attackers, the cloud is required to analyze and detect the attacking brains behind each flock of attacker bots that lead a group of coordinated attacker bots, instead of studying each user bot individually. As the computation load of a single user is never enough to cause overload on the servers. However, it is a group of users that become capable of overloading the servers.

From now on, when we say attacker, we mean an attacking brain. Each of the attacking brains may have its unique attacking type and strategies to maximize their damage through exploiting the auto-scaling feature of the cloud while minimizing their effort and, as a result, their attacking cost. There might be more than one Yo-Yo attacking brain, simultaneously forcing Yo-Yo attacks to the cloud through their own attacker bots.

Some users might have malicious volitions that act as particles of a major periodic Distributed Denial-of-Service (DDoS) attack or a Reduction of Quality (RoQ) attack. There are different types of attackers (users) because each attacking brain can perceive the cloud's scaling state. Nevertheless, all the attacker bots controlled by an attacking brain are the same types within their group.

3.2 Proposed System Formulation

3.2.1 System Overview

The two main parties of our model formulation are the cloud and the users.

$$Main\ 2\ Sides = \begin{cases} Cloud\ Provider \\ \\ Cloud\ Users \end{cases}$$

The set of users $U = \{u_1, u_2, \ldots, u_n\}$ consists of all the users using a specific service on the cloud's servers. Each i^{th} user is named u_i where i comes from the index set for users $I_U = \{1, \ldots, n\}$, thus henceforward $|U| = n$.

On the other side of this model is the cloud provider, whose aim is to serve these users to gain financial benefits for itself. These users are either normal or malicious.

$$Cloud's\,Users = \begin{cases} Normal \\ Malicious\ (Yo - Yo\ Attacker\ Bots) \end{cases}$$

The normal ones are actual human users who are actually using that service fairly on the cloud servers in order to serve their own interests (e.g., Entertainment, Communication, Business, etc.). The behavior of normal users is typically not in a particular repetitive harmonized pattern with other users.

The malicious users, on the other hand, are malicious attacker user bots who aim to perform a certain cyclic usage behavior to minimize being detected as malicious bots by the cloud's security module and at the same time, contribute to a coordinated distributed attack that, aims to force damage to the cloud in both performance and economic manners. This group of malicious attacker bots stealthily contributes to exploiting the auto-scaling mechanism of the cloud, and we call them the Yo-Yo attackers. These bots are in the set of $A = \{a_1, a_2, \ldots, a_m\}$, and as mentioned, the attacker bots set itself is a subset of the users set ($A \subseteq U$). Therefore, it is obvious that $m \leq n$. The naming of the attacker a_j comes from its index set $I_A = \{1, \ldots, m\}$, henceforth $|A| = m$.

After initial verification, at the beginning of the game, the cloud assumes all its users are normal. When the game starts, the cloud monitors all the required users' activity metrics. A user's status remains normal unless it shows suspicious behavior while using the service, making the cloud change its status from normal to suspicious or malicious. The cloud analyzes users' necessary required connection metrics to determine if each user is normal or if it is necessary to change its label from normal to suspicious or from suspicious to malicious to prevent any upcoming security risks caused by those users. All the users, including the attacker bots, are actively using the services provided by the cloud. The goal of the cloud is to set strategic security measurements to identify and defend against all malicious bots that hide within normal users by tracking and analyzing certain network parameters of the users and blocking the access of bad actors over time.

On the other side, the attacker tries to satisfy his goals. Based on its attacking brain's type and strategy, it tries to enforce maximum performance and economic damages to the cloud while keeping minimal activity to minimize the attacking costs and the probability of being caught and blocked. There are different ways for the Yo-Yo attacker to access the cloud's scaling data with different accuracy. To make this paper more compelling and to consider the most complicated situation, we assume that the Yo-Yo attacker may be able to find the auto-scaling state information accurately. In order to define each player's payoff functions in a realistic manner that properly illustrates both the profit-maximizing and seeking to minimize incurred damages cost natures of each of the players simultaneously, we need to explain their individual motivations and fears numerically so that it helps us in higher accuracy decision making.

3.2.2 The Cloud's Motivation

As we mentioned before, the motivation of the cloud is to collect maximum individual benefits from each user by providing services to them. The individual benefit $b_i(v_i)$ for each user is a function of that user's revenue value v_i that it brings to the cloud. The value of each user v_i is determined by their location, interests, age, activity amount rate, etc., and generally, any element that can affect the derived revenue targeted from each user.

The user's value metric elements definition could vary based on the type of service running on the cloud. Generally, every element that can affect the ad revenue targeted for each user can contribute to its revenue value. For instance, a streaming service could determine the user's value by location, directly affecting their ad rates, number of views, retention rates, etc. To simplify the problem, we will use numerical values for $b_i(v_i)$ and use b_i instead, so time remains the only primary variable in our model.

On the other side of the cloud provider collecting benefits from its users, there are also associated costs. The first cost that comes to mind is the default required costs from running the infrastructure to serve the users. The user i's server consumption cost at time t is determined by the total amount of computational load occupying the cloud's servers. This computational load mainly comes from the summation of $l_{u_i}(t)$, the user i's computation load on the server at time t, together with l_{c_i} the initial default constant computation load necessary for each user in order to be able to use the servers. So the user i's server cost at time t can be defined as below:

$$c_{s_i}(t) = (l_{u_i}(t) + l_{c_i})C_M$$

where C_M is the unit server's cost. We consider time as a discrete value in our calculations.

The second and third costs only appear when a Yo-Yo attack can effectively cause an unnecessary scale-up or a temporary server overload. The attacker imposes the economic damage cost when it successfully causes an unnecessary scale-up. When every scale-up happens, it takes time for the cloud's auto-scaling mechanism to trigger the scale-down mechanism; however, the cloud still needs to pay for the extra servers during the scale-up periods.

We can formulate the user i's scale-up economic damage cost contribution at time t as:

$$c_{e_i}(t) = (\frac{T_{up}C_{up}L_{up}}{\sum_{k\in I_{up}} l_k})l_{u_i}(t)$$

This function shows how much each individual user contributes to imposing the unnecessary scale-up cost to the cloud. When a scale-up happens, only the users that have been active during the scale-up thresholds are responsible for the $c_{e_i}(t)$. I_{up} indicates the active set of active users while the scale-up happened.

The economic damage cost functions show the proportion of the imposed computation load the contribution of each user l_i from the set of users I_{up} during that scale-up to the sum of computation loads on the server of the active users while scale-up ($\sum_{k\in I_{up}} l_k$). C_{up} is the unit cost of each scale-up, T_{up} indicates the total time of scale-up periods, and L_{up} is the added computation load amount after scale-up. if $L_{up} = 0$, then $c_{e_i}(t) = 0$. The last cost for the cloud appears exactly before each scale-up process. It goes back to the main reason why a scale-up happens in the first place. $r(t) = r_t$ is the response time from the server to the user for a data packet sent at the moment t.

When the initial computation load experiences a huge data burst that fully occupies its capacity, the users on that server at that time experience a higher response time, resulting in dissatisfaction. This dissatisfaction shows itself in the model as an unwanted cost that negatively affects the total profit. When the response time is in the normal range for the users, the performance damage cost is ineffective (equal to 0). It only appears when the users' response time is more significant than average.

The user i's performance damage can be shown below. L refers to the leader of this game, which we will discuss later:

$$c_{p_i}(t) = \begin{cases} 0 & 0 \leq r_t \leq r_{avg} \\ ln(\frac{r_t}{r_{avg}} c_{s_i}) & r_t \geq r_{avg} \end{cases}$$

r_{avg} is the average response time of the server to the users which is seen when the servers are in a stable condition.

As mentioned above, a cloud's server response time r_t illustrates how long it takes for cloud computers to render and respond to the user input data. The amount of response time is related to how much starvation the cloud has for CPU and memory resources, routing, etc. This hints at the fact that when the active servers are under an overload period, it takes time for the predetermined scale-up activation thresholds to trigger. During this waiting period, the users experience higher than normal r_t due to the significant processing load on the servers. This resource starvation period is the cause of the performance damage cost to the cloud.

3.2.3 The Attacker's Motivation

As done for the cloud, we also need to define the gain and loss characteristics for the attacker, which enable us to write a proper profit function. We want to take a slightly different approach to define the motivation and fear elements of the attacker. Because we look at this game through the eyes of the cloud, formulating accurate numeric representations of the gaining motivation and the fear of loss from an attacker is unknown to us. However, through observing the behaviors of different Yo-Yo attacking strategies, we define the variables equipped in a way that parametrically enables the attacker to determine a specific balance of how an attacker prioritizes economic versus performance damage, How many bots it employs in each attack, how powerful the computation overload damage capability of each bot is, and how powerful the attacker wants its Yo-Yo attack to be.

Prioritizing economic damage over performance damage means that the attacking brain engages its bots in a strategy that forces as many unnecessary scale-ups as possible. This means that it pushes the bogus traffic by its attacker bots right enough to force an overload on the existing active server's VMs, triggering an unneeded scale-up. Once the scale-up process has been completed and extra servers become online, it is best for an attacking brain with this strategy prioritization [12] to immediately stop the attack and wait till the time threshold for the scale-down to regular VM capacity triggers. After the attacking brain notices the scale-down process is being completed, It repeats the above process.

On the other hand, when an attacking brain prioritizes performance damage over economic damage, it exploits the increased response time r_t caused during the overload period before the scale-up process is triggered, it attempts to employ its attacker bots to cause maximum user dissatisfaction by creating these fake periodic overload periods for the cloud's users or even block their access for few moments. The primary purpose for an attacking brain with the above strategy prioritization is to force enough bogus traffic to induce a server overload, but long only to a limit that the bombarding traffic does not trigger the scale-up process in order to keep the servers under the maximum overload time possible while keeping the numbers of servers steady. This requires the attacking brain to have precise live knowledge of the scaling state of the cloud's servers.

In another condition, the rational option for an attacking brain is to adjust hybrid custom damage prioritization for both of these priorities based on its strategy and type. To the best of our knowledge, this is the first work that introduces a model that gives freedom to a Yo-Yo attacker to switch between prioritizing different aspects of periodic Yo-Yo attack damages. Also, it is the first work considering different types of Yo-Yo attackers.

A simple version of the proposed profit function for the Yo-Yo attacker is presented as below, where we assume the whole set of the attacker bots $A = \{a_1, a_2, \ldots, a_m\}$ is controlled by a single attacking brain:

$$H_A(t) = (T_{up}V_e + \theta(t)V_p) - \alpha \sum_{a_j \in A} p_{a_j}(t) q_{a_j}$$

As shown above, the gain of the attacker $T_{up}V_e + \theta(t)V_p)$ is determined by how valuable each of the performance damage and economic damage is for the attacking brain. This gain definition provides the required freedom for the attacker to adjust its flexibility to be suitable to perform any spectrum of damaging strategies between valuing identifying economic damage more or the performance damage.

We use V_e as an indicator value to show how much economic damage is prioritized for the attacker. or simply, the economic damage value indicator for the attacker. In addition, V_p is the indicator value to show how much performance damage is valued for the attacker. Alternatively, how vital the performance damage value is for the attacker. $\theta(t)$ indicates the performance damage success rate at time t (by receiving response time r_t), and α shows the attacking power rate.

The incurred costs element of the attacker ($\alpha \sum_{a_j \in A} p_{a_j}(t) q_{a_j}$), is defined as more straightforward than the gain element. The attacking brain attacks through its multiple attacking user bots, which can either be manipulated machines from other innocent people's resources or can be bots directly employed by the attacker to act as users on the service to damage it.

To illustrate how we calculate the attacker's attacking cost, for each bot $a_j \in A$ we multiply $p_{a_j}(t)$ the running load of j^{th} bot on the cloud servers at time t (by receiving response time r_t) by q_{a_j}, the running cost unit of j^{th} bot on the cloud servers for each bot and the sum for all attacking bots is the total attacking cost.

The summation of computing load times and its cost for all the attacker bots are limited by α coefficient, as it is an indicator of how much of the attacking power of each attacker bot is in use.

In our model, we also assume that the attacker can learn from the environment and previously observed cloud's defensive reactions to its attacks constantly and consistently aim to improve its strategy success efficiency. This helps attacking brains modify their attacking method to maximize damage by constantly adjusting their valuation of the attacking damage priorities. This means it can implement different strategies at different times based on the cloud's state and the environment's parameters. The same weighted division could also be implemented in distributing attacker bots into inflicting different attacking strategies.

Table 3.1: Table of Variables

Notation	**Interpretation**
U	Set of the users

Continued on following page

Table 3.1, continued.

Notation	**Interpretation**
u_i	The i^{th} user
A	Set of the attacker bots active among the users ($A \subseteq U$)
a_j	The j^{th} attacker bot
I_U	Index set of users ($I_U = n$)
I_A	Index set of attacker bots ($I_A = m$)
v_i *or* $v_i(t)$	Individual financial value of user i for the cloud (at time t)
$b_i(v_i)$	Financial benefit of serving user i with the value v_i for the cloud
$c_{s_i}(t)$	User i's server cost at time t
$l_{u_i}(t)$	User i's computation load on the cloud's server at time t
l_{c_i}	User i's constant necessary initial load on the server
C_M	Cloud's server's unit cost
$c_{e_i}(t)$	User i's economic damage at time t
T_{up}	Total time of scale-up periods
C_{up}	Scale-up Unit cost
L_{up}	Added computation load capacity after scale-up
$r(t) = r_t$	Server's response time to the user for a data packet sent at moment t
r_{avg}	Average response time of server to the users
$c_{p_i}(t)$	User i's performance damage cost
$H_A(t)$	The profit function of the set of attacker bots A
V_e^{π}	Economic damage prioritization value for the attacker of type π
V_p^{π}	Performance damage prioritization value for the attacker of type π
$\theta(t)$	Performance damage succeed rate at time t (by receiving response time r_t)
α	Attacking power rate
$p_{a_j}(t)$	Running load of j^{th} bot on the cloud servers at time t (and receiving response time r_t)
q_{a_j}	Running cost unit of the j^{th} bot on the cloud servers
$g_i^L(v_i, t)$	Leader's profit (Gain) function for each user i with value $v(i)$) at time t

Continued on following page

Table 3.1, continued.

Notation	**Interpretation**
$g_i^L(t)$	Simplified leader's profit (Gain) function for each user i at time t
$c_i(t)$	Accumulated costs of user i's activity on the cloud's server
$G^L(t)$	Leader's total gain function at time t for serving all the users ($\forall u_i \in U$)
X	Set of attacking brains
I_X	Index set of attacker bots ($I_X = w$)
χ_k^π	The k^{th} attacking rain of type $\pi \in \Pi$
Π	Set of attacking brains' types
π	The type of a attacking brain ($\pi \in \Pi$)
A_{χ_k}	Set of attacking bots controlled by the the k^{th} attacking brain $\chi_k^\pi \in X$
$G_{\chi_k^\pi}^L(t)$	Leader's gain function at time t for all the all the attacker bots controlled by $\chi_k^\pi \in X$
$H_k^\pi(t)$	Profit function of the k^{th} attacking brain $\chi_k^\pi \in X$ of type $\pi \in \Pi$
$U_{\chi_k^\pi}^L$	Leader's utility function by dealing with the attacking brain $\chi_k^\pi \in X$
$U_{\chi_k^\pi}^F$	Follower's utility function of type $\pi \in \Pi$
d_L	Leader's profit discount factor
d_F	Follower's profit discount factor
f_j	A vector of the attacking brain's pure strategies (Follower's policy)
l_i	A vector of the cloud's pure strategies (Leader's policy)
F	Index set of the follower's pure strategies (Follower's policy set)
L	Index set of the leader's pure strategies (Leader's policy set)

3.3 The Game Model

Now that we have defined the required elements of gains and losses for both sides of the game, we can define their profit functions, leading us to write their utility functions. Eventually, we want these functions defined to utilize them in modeling this problem as a Repeated Dynamic Stackelberg Bayesian game model.

The preliminary aspect of a Stackelberg game is knowing the leader who first picks its best strategy, and based on that, the follower aims to best respond to the leader's strategy. In this game, the cloud provider is the leader who first decides its defense strategy against the attacker, knowing that the attacker, as the follower, will try to best respond to it. attacker means an attacking brain capable of dictating the attacking strategy to its multiple attacker bots.

$$Players = \begin{cases} Cloud & (\ Leader) \\ Attacker & (\ Follower) \end{cases}$$

3.3.1 Leader's Accumulated Profit

From what we discussed in the previous section, we can write the leader's profit function in interaction with the individual user $u_i \in U$ by subtracting the costs of providing services to u_i from the gains it brings to the cloud:

$$g_i^L(v_i, t) = b_i(v_i) - (c_{p_i}(t) + c_{s_i}(t) + c_{e_i}(t))$$

By using $c_i(t)$ as the accumulated costs of the user i's activity on the cloud's server ($c_i(t) = c_{p_i}(t) + c_{s_i}(t) + c_{e_i}(t)$) we can simply write:

$$g_i^L(v_i, t) = b_i(v_i) - c_i(t)$$

To simplify this function for calculation, we'll assume $b_i(v_i)$ as a constant, not a variable ($b_i(v_i) = b_i$), so by having t as the only variable remaining, we have:

$$\Longrightarrow g_i^L(t) = b_i - c_i(t)$$

We also always assume that the leader cannot make a loss by providing services to a user, so always $b_i \geq c_i(t),\ \forall t$. As mentioned, the cloud is not playing this game against a single attacker bot. However, it plays it with an attacking brain controlling multiple attacker bots. This gives us the necessity to redefine the leader's profit function for interacting with an accumulated number of users. So the leader's total gain Function at time t for serving all the users ($\forall u_i \in U$) is:

$$G^L(t) = \sum_{i=1}^{n} g_i^L(t)$$

For calculation purposes, we can rewrite the above as:

$$\implies G^L(t) = \sum_{i=1}^{n}(b_i - c_i(t)) = nb_i - \sum_{i=1}^{n}(c_i(t))$$

3.3.2 Yo-Yo Attacking Brain

A Yo-Yo attacking brain of type $\pi \in \Pi$ is essentially an online or offline software that actively or passively controls a potentially large cluster of user attacker bots as a subset of A. There might be multiple attacking brains with their own types leading their cluster of attacker bots simultaneously to damage the cloud customers financially. The set of attacking brains is shown below:

$$X = \{\chi_1, \chi_2, \ldots, \chi_w\}$$

We can write the index set of attacking brains as $I_X = \{1, \ldots, w\}$, and as a result we can say $|X| = w$.

3.3.3 Attackers' Types

As we illustrated earlier, each Yo-Yo attacker user bot is controlled by an attacking brain and is a particle of a coordinated multi-bot periodic DDoS attack. Each attacking brain's type determines what type of Yo-Yo attacking strategy its bots use. So χ_k^π is the k^{th} attacking brain of type $\pi \in \Pi$ from the set X. The set of attacking bots controlled by the The k^{th} attacking brain $\chi_k^\pi \in X$ is shown as A_{χ_k}. Because we considered that each Yo-Yo attacker $a_j \in A$ is controlled by one, and only one attacking brain $\chi_k^\pi \in X$, we can simply claim that:

$$A = \bigcup^{\chi_k \in X} A_{\chi_k}$$

This means that all of the attacker bots involved in each Yo-Yo attack are controlled by one and only one attacking brain of type $\pi \in \Pi$, or more mathematically accurately:

$$\forall a_j \in A, \exists \chi_k \in X \ s.t. \ a_j \in A_{\chi_k}$$

3.3.4 Leader's Utility Function

Now that we have defined the types for each attacker $[)a_j \in A_{\chi_k} \subseteq A$ controlled by attacking brain χ_k, we can define the game between the cloud as the leader facing the k^{th} attacking brain with type $\pi \in \Pi$ that controls the set of attacker bots A_{χ_k} as the follower of the game.
To write the leader's gain function at time t in a game against the attacking brain $\chi_k^\pi \in X$ that controls the attackers of the set A_{χ_k}, we have:

$$G^L_{\chi_k^\pi}(t) = \sum_{a_z \in A_{\chi_k}} g^L_{a_z}(t)$$

To make it usable for calculation, we can write this as below:

$$\Longrightarrow G^L_{\chi_k^\pi}(t) = \sum_{a_z \in A_{\chi_k}} (b_{a_z} - c_{a_z}(t)) = |A_{\chi_k}| b_{a_z} - \sum_{a_z \in A_{\chi_k}} (c_{a_z}(t))$$

After finding the gain (Profit) function of the leader, we use it to write the leader's utility function by dealing with the attacking brain $\chi_k^\pi \in X$:

$$U^L_{\chi_k^\pi} = \sum_{t=0}^{t_{current}} d_L G^L_{\chi_k^\pi}(t)$$

$d_L = \gamma^t$is a discount factor($0 < \gamma < 1$) that multiplies the gain function in order for us to show we prioritize weighting the most recent gains more than the older ones. In order to use the above function for calculation, we simplify it:

$$U^L_{\chi_k^\pi} = \sum_{t=0}^{t_{current}} \sum_{a_z \in A_{\chi_k}} \gamma^{\frac{1}{t}} g_{a_z}(t) \Longrightarrow U^L_{\chi_k^\pi} = \sum_{t=0}^{t_{current}} \sum_{a_z \in A_{\chi_k}} \gamma^{\frac{1}{t}} (b_{a_z} - c_{a_z}(t))$$

3.3.5 Follower's Utility Function

Previously, we defined the accumulated profit function for all attacker bots A. In this section, we want to be more specific. Now we specifically write the profit function (Follower's profit function) for the k^{th} attacking brain $\chi_k^\pi \in X$ of type $\pi \in \Pi$, its profit function could be defined as:

$$H_k^\pi(t) = (T_{up}V_{e_k}^\pi + \theta_k(t)V_{p_k}^\pi) - \alpha_k \sum_{a_z \in A_{\chi_k}} p_{a_z}(t)q_{a_z}$$

$V_{e_k}^\pi$ is the performance damage value for the attacker. In other words, it is an indicator value to show how much causing performance damage is valued by the attacking bot χ_k^π.
Similarly, $V_{p_k}^\pi$ indicates how much forcing Performance damage to the cloud is valued by the attacking brain $\chi_k^\pi \in X$. Also for the k^{th} attacking brain profit function, $\theta_k(t)$ represents its performance damage success rate at time t (by receiving response time r_t), and α_k is the attacking power rate.

In order to write the follower's profit function, the expenses of performing such a multibot attack are then deducted from its income value. To calculate the attacking cost for an attacking brain, we multiply $p_{a_j}(t)$, the processing load of the j^{th} attacker bot (a_j)on the cloud servers at time t (by receiving response time r_t), by q_{a_j} the running cost unit of the j^{th} attacker bot on the cloud servers.

So, to write the attacking brain's total expenses, we multiply the attacking power rate by The summation of individual attacking costs for running all the attacker bots ($\sum_{a_z \in A_{\chi_k}} p_{a_z}(t)q_{a_z}$).
Now that we have fully explained the profit function for the attacker, we can write the follower's utility function as below:

$$U_{\chi_k^\pi}^F = \sum_{t=0}^{t_{current}} d_F H_k^\pi(t)$$

$d_F = \mu^t$ is the discount factor ($0 < \mu < 1$) used to put higher numerical priority on the most recent profits than the older ones. For an easier calculation process, we simplify the above formula:

$$U_{\chi_k^\pi}^F = \sum_{t=0}^{t_{current}} \mu^t H_k^\pi(t) \implies U_{\chi_k^\pi}^F = \sum_{t=0}^{t_{current}} \mu^t \big((T_{up}V_{e_k}^\pi + \theta_k(t)V_{p_k}^\pi) - \alpha_k \sum_{a_z \in A_{\chi_k}} p_{a_z}(t)q_{a_z}\big)$$

Now that we have shown both players' utility functions, we can fully start implementing backward induction to solve our Repeated Dynamic Stackelberg Bayesian game modeling of the Yo-Yo attacks.

3.4 Assumptions

Centralized Cloud Entity: In our model, "the cloud" is conceptualized as a singular, cohesive entity. It possesses comprehensive capabilities and authority, ensuring that all requisite decisions related to scaling strategies, monitoring user activity metrics, imposing restrictions on malicious entities, and orchestrating defense mechanisms are centralized. Such decision-making processes can be delegated to components within the cloud, including but not limited to load balancers, virtual machines (VMs), and other infrastructure components.

User Grouping and attacker brain Dynamics: Users exhibiting synchronous behavioral patterns, especially during critical intervals—precisely when the cloud server's auto-scaling mechanism threshold timer is about to be triggered—are categorized as the constituents of a singular attacking entity termed as the "attacking brain". This brain operates as the strategic nucleus, guiding the actions of its associated users, which we analogize as its "soldiers".

Unique Attacker brain Typology: Distinctiveness among attacking brains is demarcated based on their type. Should an attacking brain bifurcate its associated entities (bots) to execute varied strategic profiles, each individual profile is regarded as a unique attacking brain. This abstraction assigns each strategy profile its own distinct type and associated characteristics.

Attacker's Scaling State Insight: We operate under the assumption that the attacker, regardless of its sophistication or type, possesses the capacity to ascertain the precise scaling state of the cloud servers at any given juncture. Furthermore, it can dynamically discern patterns within the cloud's scaling strategy, allowing it to adapt and recalibrate its offensive techniques in real-time.

Chapter 4

Formulating Yo-Yo Attack as Repeated Stackelberg Bayesian Security Game Model

After thoroughly defining the game elements in the last chapter, we thoroughly write the Repeated Dynamic Stackelberg Bayesian security game modeling formulation of Yo-Yo attacks in this chapter. After considering the follower's best response to its implemented strategy, we implement backward induction to find the leader's optimal defense strategy for the cloud provider. Our final goal in formulating this problem is finding the cloud's optimal defense strategy against all Yo-Yo attackers using backward induction.

4.1 Repeated Dynamic Stackelberg Bayesian Security Game

Initially, we want to review our game model's overall structure. First of all, our game is dynamic. By dynamic, we mean that both players can make optimal decisions after following and analyzing each other's actions. Also, by the definition we provided for our game environment, we can see that the interaction between the players is intrinsically dynamic. Because the attacker actively observes the scaling state of the cloud's servers and picks the optimal attacking strategy accordingly to enforce the maximum damage to the leader, and on the other side, the cloud as the leader is actively watching all the users' activity patterns and implements security measurements to find out and block any suspicious activity by the users. As we see in the mathematical formulation of this problem, we can notice that this model is a singular one-off game that repeats over time. In each iteration of this game, the players can investigate the outcomes they have already observed and learned from their opponent and then adjust their parameters to get a better outcome when playing next time.

Secondly, This game is Stackelberg. This comes from the fact that there is inherently a sequential nature in the decision-making process of this game. This means that the attacker launches a successful Yo-Yo attack, most likely after it is informed about the scaling state of the cloud. This eventuates that in this game, the cloud as the leader is first executes its defense and scaling strategy, and based on these policies, the attacker as the follower observes it and, based on that, picks the best response strategy.

Thirdly, this game is Bayesian. to have a more accurate and practical formulation of this problem, we consider the fact that we previously explained how different the inference of different attacking Brains from the auto-scaling state of the cloud can be. Considering freedom for defining different attackers' types in our formulation gives our model more credibility as it enables the cloud to detect and defend against a broader spectrum of periodic attacks that focus on the auto-scaling vulnerability of the cloud. Also, it keeps the doors open for the leader to add any new type of Yo-Yo attacks that have not been previously observed and then define it as a new type of attacker in the attacking types set.

4.2 Players Policies

4.2.1 Leader's Policy Set

In practice, for every user active on the cloud's servers, the cloud can offer different behaviors versus its users. If the cloud thinks a user is normal, it keeps the user's connection to the servers intact. However, if the cloud thinks a user is malicious, it immediately blocks that user's access to the servers. In other conditions, if the cloud is suspicious of a user and is not sure if the user is legit or malicious, it considers it as a suspicious user and can either make that user pass further authentication methods like CAPTCHA, etc., or it can put suspicious users in a honeypot in order to learn their behavior patterns more specifically by manipulating their response time to disrupt their attacking mechanism and then observe those users reaction to those disruptions. The leader's pure policy set is shown below:

$$Leader's\ Strategies = \begin{cases} OK\ to\ go & (Normal\ Users) \\ Inject\ delay\ or\ I'm\ Not\ Robot\ test & (Suspicious\ Users) \\ Block & (Malicious/Bad\ Users) \end{cases}$$

4.2.2 Follower's Policy Set

On the other hand, the attacking Brain, as the follower in the game, chooses its action from its strategy set. Based on the follower's perception of the cloud's scaling state and its attacking type and strategies, the attacking Brain decides whether to attack the cloud or not attack it. The follower's pure policy set is shown below:

$$Follower's\ Strategies = \begin{cases} Attack \\ No\ Attack \end{cases}$$

Now that we have figured out the leader's and follower's pure strategies individually, we can discuss finding the cloud's optimal solution using the backward Induction.

4.3 Backward Induction

We analyze optimal strategies for both the cloud provider and the attacking brains through the lens of backward induction. We commence by defining the attackers' optimal response to a given, fixed strategy of the cloud provider. This response is subsequently integrated into the cloud provider's optimization problem, guiding it toward its optimal detection and interdiction strategies.

This suggests that the cloud provider projects the attacking brain's optimal responses to its observed defensive maneuvers. This projection is incorporated into its optimization problem to dictate the ideal detection strategy.

Consider L and F as index sets signifying the pure strategies of the cloud provider (Leader) and the attacking brain (Follower), respectively. Let l symbolize a vector of the cloud provider's pure strategies (Leader's policy), and f signifies a vector of the attacking brains' pure strategies (Follower's policy). In this context, l_i demonstrates the frequency of the leader deploying pure strategy i from its policy set, indicating how the cloud provider interacts with each user (providing a standard service, blocking, inducing delay, CAPTCHA verification, etc.). Similarly, f_j delineates the frequency of the attacking brain pure strategy j from its policy set, thus portraying whether an attack is launched or deferred at the time t.

4.3.1 Follower's Optimization Problem

First, let us fix the leader's policy to a particular policy l. When the follower observes the leader's vector of pure strategy l, in order to best respond to l, it needs to solve the following linear programming optimization problem:
Having fixed the leader's (cloud provider's) policy to a specific strategy denoted as l, the follower (attacking brain), upon observing this strategy vector l, has to respond optimally. To facilitate this, the follower must solve the subsequent linear programming optimization problem:

$$max \sum_{j\in F}\sum_{i\in L} U^F_{ij} f_j l_i S.t. \sum_{j\in F} f_j = 1 f_j \in [0,1], \quad \forall j \in F$$

In this scenario, the attacker's best response $f_j(l)$, given the known fixed strategy l of the cloud provider [30], should result in a non-negative utility for the attacker. The constraint encapsulates this:

$$f_j \times \sum_{i\in L} U^F_{ij} \times l_i \geq 0, \quad \forall j \in F$$

Furthermore, $f_j(l)$ being the best response, any deviation $(1 - f_j)$ would imply a utility loss for the attacker. Hence, the constraint below is also required:

$$(1 - f_j) \times \sum_{i\in L} U^F_{ij} \times l_i \leq 0, \quad \forall j \in F$$

The value f_j encapsulates the attacker's optimal pure strategy in response to the leader's observed strategy l_i. Consequently, f_j can only be 0 or 1. If the best response f_j equals 1, indicating that the attacker opts to attack, the deviation from f_j would be 0, yielding an attacker utility of 0. Intuitively, this means the attacking brain neither gains nor loses anything, as no attack was executed. Alternatively, if the attacker's best response f_j is 0, the deviation from f_j would be 1, implying that the attacker chooses to attack. In this case, the attacker's utility will be consistently negative, symbolizing an unsuccessful attack due to detection by the cloud provider and the costs incurred to initiate the attack.

4.3.2 Leader's Optimization Problem

Let us move now to the leader's side. The cloud provider, knowing that the attacking brain will play its best response $f_j(l)$ to every leader's strategy l, incorporates this knowledge into its optimization problem to determine the solution l that maximizes its payoff. Thus, the cloud has to solve the following problem:

$$max \sum_{i \in L} \sum_{j \in F} U^L_{ij} f_j(l) l_i$$

$$S.t. \sum_{i \in L} l_i = 1, \quad l_i \in [0, 1], \quad \forall i \in L$$

Switching perspectives to the cloud provider, it considers that the attacking brain will execute its optimal response $f_j(l)$ to each of its strategies l. By integrating this knowledge into its own optimization problem, the cloud provider determines the strategy l that yields the maximum payoff. Thus, the provider must solve the following problem:

$$max \sum_{i \in L} \sum_{j \in F} U^L_{ij} f_j(l) l_i$$

$$S.t. \sum_{i \in L} l_i = 1, \quad l_i \in [0, 1], \quad \forall i \in L$$

This optimization problem allows the cloud provider to identify a strategy that maximizes its payoff, considering the likely best responses from the attacking brain.

4.4 Mixed-Integer Quadratic Programming (MIQP) Problem

Given any optimal mixed strategy $f_j(l)$, it logically follows that all pure strategies within its support are also optimal. This can be proven simply via a contradiction argument. If we assume there is a pure strategy within the support that is not optimal, then there exists another strategy that can offer a higher utility. Nevertheless, this contradicts the premise that $f_j(l)$ was an optimal mixed strategy since incorporating this new strategy would yield a higher utility. Subsequently, we can focus on the attacking brain's optimal pure strategies, which always exist, and represent them using binary variables.

To further refine the cloud provider's decision-making, we include the probability distribution p^π pertaining to each type $\pi \in \Pi$ of attackers into the cloud's optimization problem. For instance, suppose the types of Yo-Yo attackers are categorized into sophisticated, intermediate, and novice, each with a distinct likelihood p^π of occurring. The cloud provider, aware of these probabilities, can adjust its strategies accordingly. The following section will detail how the cloud provider can practically compute p^π. With all this information at hand, the cloud provider's problem evolves into:

$$\max_{l,f} \sum_{\pi\in\Pi}\sum_{i\in L}\sum_{j\in F} p^\pi U_{ij}^{L,\pi} l_i f_j^\pi$$

$$S.t. \sum_{i\in L} l_i = 1$$

$$\sum_{j\in F} f_j^\pi = 1, \quad \forall \pi \in \Pi$$

$$(1 - f_j^\pi) \times \sum_{i\in I_L} U_{ij}^F \times l_i \leq 0, \quad \forall j \in F, \pi \in \Pi$$

$$f_j^\pi \times \sum_{i\in I_L} U_{ij}^F \times l_i \geq 0, \quad \forall j \in F, \pi \in \Pi$$

$$l_i \in [0,1], \quad \forall i \in L$$

$$f_j^\pi \in \{0,1\}, \quad \forall j \in F, \pi \in \Pi$$

The aforementioned maximization problem constitutes an integer program with a non-convex quadratic objective. In this formulation, the first and sixth constraints enforce the feasibility of a mixed policy for the cloud provider. In contrast, the second, third, and seventh constraints ensure a feasible pure strategy for the attacking brain. The fourth and fifth constraints guarantee the best response $f_j(l)$ to be optimal for the attacker concerning utility gain. The sixth constraint also dictates the attacking brain's action vector to be a pure distribution over F.

4.5 Convert MIQP to a Mixed-Integer Linear Programming (MILP) Problem

The final stage of this chapter involves transforming this Mixed-Integer Quadratic Programming (MIQP) problem into a Mixed-Integer Linear Programming (MILP) problem. This is accomplished by eliminating the nonlinearity of the objective function, which can be achieved by substituting the value of $l_i \times f_j^\pi$ with a new variable z_{ij}^π.

The product of the decision variables is removed from the objective function and replaced with the new variables z_{ij}^π, hence transforming the problem into a linear one. These new variables are constrained to be between 0 and 1, consistent with the feasible range of the original decision variables.

Additionally, constraints are added to ensure that the new variables z_{ij}^π accurately represent the product of the original decision variables l_i and f_j^π. As a result, the problem preserves its original meaning while eliminating the quadratic term from the objective function, rendering it a MILP problem. Consequently, the revised MILP problem is as follows:

$$\max_{l,f} \sum_{\pi\in\Pi} \sum_{i\in L} \sum_{j\in F} p^\pi U_{ij}^{L,\pi} z_{ij}^\pi$$

$$S.t. \sum_{i\in L} \sum_{j\in F} z_{ij}^\pi = 1, \quad \forall \pi \in \Pi$$

$$f_j^\pi \le \sum_{i\in L} z_{ij}^\pi \le 1, \quad \forall j \in F, \pi \in \Pi$$

$$\sum_{j\in F} f_j^\pi = 1, \quad \forall \pi \in \Pi$$

$$(1 - f_j^\pi) \times \sum_{i \in L} U_{ij}^F \times z_{ij}^\pi \leq 0, \quad \forall j \in F, \forall \pi \in \Pi$$

$$f_j^\pi \times \sum_{i \in L} U_{ij}^F \times z_{ij}^\pi \geq 0, \quad \forall j \in F, \forall \pi \in \Pi$$

$$z_{ij}^\pi \in [0, 1], \quad \forall i \in L, j \in F, \pi \in \Pi$$

$$f_j^\pi \in \{0, 1\}, \quad \forall j \in F, \pi \in \Pi$$

These steps effectively transform the original MIQP into an equivalent MILP, making it more manageable and easier to solve while still respecting the initial conditions and constraints of the original problem.

4.6 Learning-Based Attackers' Type Recognition and Defense Mechanism

Previously, we delved into the intricacies of Yo-Yo attacks targeting cloud servers and devised a framework modeled on the Repeated Dynamic Stackelberg Bayesian game. This analytical construct can be deconstructed into three salient phases:

(1) **Dynamic Bayesian Stackelberg Game:** As elucidated in the preceding chapter, this phase aims to determine the optimal probability distributions of the cloud's strategic defense and attack detection mechanisms in relation to each potential user or attacker action.

(2) **Defense Module Mechanism:** Triggered when an anomaly suggests a user might be under the malignant influence of a Yo-Yo attacking brain, this phase endows the cloud manager with a gamut of defensive strategies. Options range from outright blocking the suspected user, routing them through CAPTCHA validations, inducing intentional lags in data packet responses to disorient their understanding of the cloud's scalability, and funneling them towards honeypots to decode their attack modalities.

(3) **Learning-Based Attackers' Type Recognition:** This phase underscores the essence of proactive monitoring. By continually scrutinizing user activities, it aims to discern the nuances in attackers' reactions to varied response times. By collating and examining data regarding their

malevolent strategies, the goal is to gain insights into the motivations and methodologies of the attackers, mainly when governed by an attacking brain.

It is pivotal to understand that these phases are not static but iterate over each discrete time unit, denoted as $[t_1, t_2]$. The amassed data undergoes rigorous analysis via a one-class SVM classifier aimed at discerning the nature and taxonomy of Yo-Yo attackers, thereby enhancing our understanding of their probability distributions. This invaluable intelligence subsequently informs the Dynamic Bayesian Stackelberg game during the succeeding time unit $(x + 1)$, enabling recalibration and optimization of the Cloud's strategy for detection load distribution. A pertinent caveat is the cyclical nature of the Bayesian Stackelberg game, necessitating its repetition for every discrete-time window unit. Instigating auxiliary phases hinges on the outputs derived from the game phase. To elucidate, should no user be flagged as a potential threat during a specific time frame, the subsequent steps for that particular duration become redundant.
In the ensuing sections, we shall delve deeper, offering a granular examination of each phase within the Repeated Stackelberg game framework.

4.6.1 The Defense Mechanism Module

Central to our discussion is the defense Mechanism Module, a pivotal construct that epitomizes the cloud's defensive apparatus.

This module is summoned into action once a user is earmarked, ostensibly governed by a Yo-Yo Attacking brain. The flexibility of our model is its hallmark. The cloud's responses are not only contingent upon the nature of the suspected attack. However, they are also tailored based on the services it offers and the vulnerabilities it is susceptible to. This nuanced approach allows for a more dynamic defense, catering to the specific vulnerabilities of different cloud services. For instance, a cloud service dedicated to financial transactions may prioritize different defense mechanisms than one hosting public web content.

Stratified Defense Responses

(1) **Blocking:** Foremost in the arsenal is the straightforward act of barring access. Once a user is unequivocally identified as a malevolent bot, the optimal recourse is immediate blocking, safeguarding the cloud infrastructure from any impending harm.

(2) **Verification Measures:** When suspicion is cast but conclusive proof eludes, softer verification tools are employed. CAPTCHA serves as a prime example. However, in an era where authenticity is paramount, advanced measures, such as 2-factor authentication, become indispensable. This could entail transmitting a one-time code to a user's registered mobile number or email, ensuring an additional layer of validation.

(3) **Delay Injection:** Tailored especially for countering the nuances of Yo-Yo attacks, delay injections serve a dual purpose. They not only befuddle attackers, rendering them unable to accurately gauge the cloud's auto-scaling algorithm but also, when deployed within honeypots, help discern how these attackers recalibrate based on fluctuating response times.

(4) **Honeypots:** A sophisticated decoy, honeypots are deployed to lure attackers, affording an observational vantage point to understand their modus operandi. Every interaction with a honeypot is intrinsically unauthorized, rendering all traffic to it malicious. The objective is clear: simulate an authentic environment, prompting attackers to reveal their strategies, thus amassing invaluable intelligence.

To explain the differences between low-interaction vs. high-interaction Honeypots, The efficacy of a honeypot is determined by its interaction level. Low-interaction honeypots offer limited interaction, generally logging basic information about the attacker. Conversely, high-interaction honeypots are more immersive, allowing attackers to engage with a seemingly genuine system. Such an environment is ripe for studying diverse attack vectors, even those unanticipated.

Given our overarching aim—to meticulously study attacker behavior to extrapolate probability distributions over their types—the choice of high-interaction honeypots is self-evident. Their unbiased nature, making no preordained conjectures about attacker behavior, makes them adept at recording a spectrum of malicious endeavors, even those that veer off the beaten path. This empowers us to not only document but also analyze a gamut of attack paradigms, including those hitherto unknown.

4.6.2 Learning-Based Attackers' Type Recognition

Post data accumulation via the defense Module's honeypots, an astute classification methodology is requisite to dissect this data, aiming to deduce the probability distributions across varied attacker profiles.

Our preference leans towards the one-class Support Vector Machine (SVM). Originally conceived as an augmentation of the conventional binary SVM classifier [31], the one-class SVM delineates itself with a unique approach. It seeks a hyperplane, which, in mathematical parlance, is the decision boundary. This boundary judiciously distances the lion's share of data points from the origin, effectively labeling data points beyond this demarcation as anomalies or outliers. This architecture equips the decision function with the acumen to appraise new data, discerning whether it aligns with or diverges from a specific data pattern assimilated during its training epoch, a process colloquially termed novelty detection.

Three pivotal observations undergird our inclination towards the one-class SVM:

(1) **Unsupervised Nature:** The one-class SVM, in its essence, is an unsupervised classifier. This implies it operates without heavily relying on preliminary information or the need for pre-tagged class labels in the dataset under scrutiny.

(2) **Multifaceted Classification:** Its innate ability to seamlessly handle multi-class data classification positions it as a prime choice for our conundrum, where the landscape is populated with a diverse range of attacker archetypes.

(3) **Novelty Detection Prowess:** The one-class SVM's core competency lies in its novelty detection. This trait empowers it to unearth nascent, uncharted attack vectors. The SVM gauges the congruence or divergence of novel data concerning this 'normal' dataset by contextualizing established attack modalities as normative behavior. This capability is invaluable, especially when the objective is to identify and combat emerging threats preemptively.

Imagine a scenario wherein the classification system has previously discerned two distinct types of attackers. Should a novel attacker manifest with attributes that diverge from the archetypes of these pre-identified attacker types, this new manifestation would be interpreted as a nascent attack variant, potentially imperiling the cloud system.

To elucidate this formally, let us define $x = (x_1, x_2, \ldots, x_n)$ as the feature vector encapsulating a myriad of attack attributes, such as source and destination IP addresses, employed protocols, host-names, the geographical origin of the attack, among others, all collated by the honeypot apparatus. The one-class SVM, in its crux, translates the classification conundrum into an objective function minimization problem delineated as:

$$\begin{cases} \min\limits_{\omega,\xi_i,\rho} \frac{1}{2}\|\omega\|^2 + \frac{1}{vn}\sum\limits_{i=1}^{n}\xi_i - \rho \\ s.t: \\ (\omega.\phi(x_i)) \geq \rho - \xi_i & \forall i = 1,\ldots,n \\ \xi_i \geq 0 & \forall i = 1,\ldots,n \end{cases}$$

Here, n represents the training set's magnitude, while ω and ρ correspond to the hyperplane's normal vector and bias term, respectively. The transformation function, symbolized by $\phi(.)$, and facilitated by the kernel function, projects data into a more expansive dimensional ambit. The slack variables, represented as $(xi_i \in \xi_n$, permit certain data instances to nestle within the margin, thereby inoculating the SVM classifier against over-fitting in the face of noisy data. Of pivotal import is the regularization parameter, v, which informs the resultant solution's contour by stipulating: (1) a cap on outlier fractions and (2) a lower bound on the volume of training tuples functioning as support vectors. Increasing v engenders a more expansive soft margin, augmenting the likelihood of training data breaching conventional borders. This problem can be astutely tackled by leveraging the Lagrange multipliers technique, leading to the decision function $f(x)$ materializing as:

$$f(x) = sgn((\omega.\phi(x_i)) - \rho) = sgn(\sum_{i=1}^{N}\alpha_i k(x, x_i) - \rho)$$

Within this formulation, the kernel function $k(x, x_i)$ could adopt various forms, including linear, polynomial, Gaussian, or sigmoid, delineated as:

$$K(x_j, x_i) = \begin{cases} x_i, x_j, linear \\ (\gamma.x_i.x_j + c)^d, polynomial \\ \exp(-\gamma.|x_i - x_j|^2), gaussian\ radial\ basis \\ \tanh(\gamma.x_i.x_j + c), sigmoid \end{cases}$$

Gleaning insights from the classification outcome, the cloud calculates the probability p^π for each attacker profile $\pi \in \Pi$, deduced as:

$$p^\pi = \frac{Number\ of\ observations\ classified\ as\ "\pi"}{Total\ number\ of\ observations}$$

In culmination, this intelligence is retrofitted into the Bayesian Stackelberg game (as expounded in the preceding chapter), furnishing the game with updated probability distributions over attacker types. This iterative feedback optimally refines the game's detection load distribution across user subsets.

Chapter 5

Experimental Simulations and Results Analysis

In this chapter, we delve into the experimental environment, methodologies, and the subsequent results derived from our research. We aim to provide a comprehensive understanding of the performance and implications of our proposed solution compared to the state-of-the-art Yo-Yo mitigation systems. The experiments are designed in a way to show the different reactions of the AD, ADAI, and MDAI TASD defense solutions compared to our mitigation technique solution while facing three different types of Yo-Yo auto-scaling exploitation attack scenarios.

5.1 Simulation Experiment Setup

In this section, we will define every necessary element for creating the simulation environment. We will discuss our experimental simulation setup hardware and software details, users' quantity and behavior, different attacker types, and their type's probability distribution. Also, we discuss the state-of-the-art Yo-Yo attack mitigation mechanisms. After that, we will define our mitigation method. After discussing all the required knowledge. In the next section, we perform the experiment simulations to analyze the effectiveness and advancements of our provided solution compared to the most recent related works.

5.1.1 Simulation Environment

In our pursuit to model and analyze the intricate dynamics of the different types of Yo-Yo attacks on cloud systems using Bayesian Stackelberg game theory, it became imperative to devise a bespoke simulation environment. This necessity stemmed from the inherent limitations of existing platforms, such as AWS and CloudSim. While these environments offer robust features, they fall short of offering the granular control and specific functionalities essential for our research. For instance, AWS, despite its scalability [32, 33] and comprehensive infrastructure, restricts access to critical monitoring and threshold adjustment capabilities necessary for our simulations. Similarly, CloudSim, though widely used for cloud simulations, imposes constraints on defining auto-scaling criteria, and its updates post-version 3.0.3 have lacked the consistency required for our advanced modeling needs.

To effectively simulate the diverse range of Yo-Yo attack scenarios and test various defense mechanisms, our team decided to develop our own cloud simulation environment using Python. This approach allowed us to overcome the accessibility and functionality limitations encountered with AWS and CloudSim. By custom-building our simulation environment, we achieved a dual objective: We replicated the comprehensive feature set of these platforms while gaining the flexibility to tailor every aspect of the simulation to our specific research needs. This included the ability to define intricate detection mechanisms and auto-scaling features that are pivotal in modeling and mitigating Yo-Yo attacks. Our environment, therefore, represents the best of both worlds – combining the robustness of AWS and CloudSim with the customized control and specificity required for our cutting-edge research.

The decision to develop our own cloud client simulator in Python has been instrumental in accurately replicating the cloud environment used in [20], but also with more sophisticated features in hosting both the normal users and attackers simultaneously during the simulation. This unique approach not only facilitates a more realistic simulation of cloud interactions but also allows for an in-depth exploration of Yo-Yo attack dynamics under varied conditions. Our environment's elasticity and customizability enable us to meticulously monitor and adjust system parameters in real-time, a capability crucial for understanding and countering these complex cyber threats. Consequently, our custom simulation environment stands as a testament to innovative problem-solving and is a cornerstone in advancing the field of cloud security and game theory-based attack modeling.

5.1.2 Yo-Yo Attackers Types

Introducing 3 different Sample Attacker Types Strategies:

(a) **Yo-Yo Attacker Type 1**

This type of attacker is the most similar to all the previous works written about the yo-yo attack. The attacker pushes a DDoS attack data burst. Once the attacker ensures the attack has successfully forced a scale-up, it stops it and waits for the cloud to scale back down to its default capacity. Then, the attacker does the same thing over and over. This type's primary focus is causing maximum oscillation in the server's processing capacity to induce economic and performance damages.

(b) **Yo-Yo Attacker Type 2**

This type of attacker's primary focus is forcing maximum performance damage for the cloud; however, causing a scale-up is not a priority for this type. This attacker type's most desirable attacking strategy is to push its attack force long enough to cause server overload but not too long to trigger the overload interval. For example, suppose the cloud manager has set the scaling criteria to trigger a scale-up process after two consistent minutes of utilization over 70%. In that case, the attacker of type 2 prefers to push maximum data burst but for a period shorter than the 2-minute threshold (e.g., 1.9 min) in order to not trigger a scale-up and keep the servers in an overload state, which results in the increased response time and maximum dissatisfaction for the normal active users on the server and eventually a heavy performance damage for the service provider. Once the scale-up threshold has restarted before triggering the scale-up, The attacker does the same process again.

(c) **Yo-Yo Attacker Type 3**

This type, similar to type 2, takes a more radical approach to enforcing a yo-yo attack, but in a different way. This type of attacker's primary focus is forcing maximum economic damage on the cloud by making the cloud employ extra servers for the maximum amount of time. This type's primary focus is keeping the cloud's server in the scale-up state for the maximum duration. Nevertheless, once the first attack wave of the attacker type 3 triggered a scale-up, waiting for the servers to scale down is not a priority for this type like it is for the type 1 attacker. This attacker type's most desirable attacking strategy is to push its attack force long enough to cause server scale-up. Once the scale-up occurs, the attacker stops attacking, but this pause is not too long to trigger a scale-down. For example, suppose the cloud manager has set the scaling criteria to trigger the scale-down process after two consistent minutes of utilization below 30%. In that case, the attacker of type 3 prefers to stop push maximum data burst but only waits for below the 2-minute threshold (e.g., 1.9 min) in order to not trigger a scale-down and keep the servers in a scaled-up state, which results in increased scale-up time and maximum costs for paying the extra unnecessary processing infrastructure. The attacker does the same process over again.

5.1.3 The Probability Distribution over Attackers' Types

5.1.4 State-of-the-Art Yo-Yo Attack Mitigation Approaches

Trust-based Adversarial Scanner Delaying (TASD) mechanisms are state-of-the-art approaches specifically designed to detect and mitigate the Yo-Yo attacks.

The AD TASD offers a baseline defense with its straightforward trust value reduction approach, ADAI TASD enhances this model by introducing dynamic trust value adjustment to reduce false positives, and MDAI TASD builds upon these strategies by implementing a more aggressive response to suspicious activities. Collectively, these mechanisms form a layered defense strategy, ensuring robust security in cloud auto-scaling environments against sophisticated Yo-Yo attacks.

AD TASD

AD TASD (Additive Decrease Trust-based Adversarial Scanner Delaying) operates on each instance of a web application within an auto-scaling group. As the system scales up, the distributed TASD service on each new instance functions independently as a defense mechanism. It primarily focuses on detection and mitigation, where the attack detection module categorizes each user request. Suspicious requests are forwarded to a secondary auto-scaling group, where they encounter a web service that imposes a two-second delay, or in the case of malicious requests, they are dropped. The users marked as normal will be forwarded to the main auto-scaling module. Algorithms 1 and 2 show the pseudo-codes for detection and mitigation methods of AD TASD.

Algorithm 1 Pseudo code for the AD trust value detection Method

```
for each request r do
    if the user u in request r is not in trust value db then
        Add user u with default trust value $T_{init}$ to trust value db;
    end if
    if Auto-scaling in Scale-out then
        $N_i \leftarrow$ number of requests for each users;
        S $\leftarrow$ k users with top t request numbers;
    end if
    if Auto-scaling in Scale-in then
        for each user ui in S do
            $N_i' \leftarrow$ number of requests for $u_i$;
            if $N' - N > M$ then
                Update trust value of user $u_i$ : $T_i = T_i - 1$;
            end if
        end for
    end if
end for
```

Algorithm 2 Pseudo code for the AD trust value mitigation method

```
for each request r do
    if the trust value of request $T_i < T_{suspicious}$ then
        Add user u to ALB forwarding rule;
    end if
    if the trust value of request $T_i < T_{malicious}$ then
        Add deny rule for user u in VPC ACL;
    end if
end for
```

ADAI TASD

ADAI TASD (Additive Decrease/Additive Increase Trust-based Adversarial Scanner Delaying) addresses the false-positive error by dynamically updating the trust value of users. It increases the trust value of a user after a certain interval, balancing between detecting malicious activities and avoiding mislabeling normal users. Compared to the AD method, the ADAI method aims to reduce false positives by adapting to user behavior by offering a more nuanced approach to managing user trust values. Algorithm 3 shows the pseudo code for the ADAI TASD method.

Algorithm 3 Pseudo code for the ADAI trust value mitigation method

```
for each request r do
    if the trust value of request $T_i < T_{suspicious}$ then
        if $t_{forward}$ of $u_i > t_{release}$ then
            Update trust value of user $u_i$ : $T_i = T_i + 1$);
        else
            Add user s to ALB forwarding rule;
        end if
    end if
    if the trust value of request $T_i$!‘$T_{malicious}$ then
        if $[)t_{block}$ of $u_i > t_{release}$ then
            Update trust value of user $u_i$ : $T_i = T_i + 1$;
        else
            Add deny rule for $u_i$ in VPC ACL;
        end if
    end if
end for
```

MDAI TASD

MDAI TASD (Multiplicative Decrease/Additive Increase Trust-based Adversarial Scanner Delaying) incorporates a multiplicative decrease in trust values, allowing for a more aggressive response to potential threats. This method is effective in swiftly escalating defenses against suspicious activities.. MDAI TASD's strategy of combining a multiplicative decrease with an additive increase ensures that the system can quickly respond to potential threats, while still allowing for the rehabilitation of users who may have been falsely categorized as malicious. This dual approach ensures that the system remains both secure and fair, adapting dynamically to the evolving patterns of user behavior and potential threats. Algorithms 4 and 5 show the pseudo-codes for detection and mitigation methods of the MDAI.

Algorithm 4 Pseudo code for the MDAI trust value detection method

```
for each request r do
    if the user u in request r is not in trust value db then
        Add user u with default trust value $T_{init}$ to trust value db;
    end if
    if Auto-scaling in Scale-out then
        $N_i \leftarrow$ number of requests for each users;
        S $\leftarrow$ k users with top t request numbers;
    end if
    if Auto-scaling in Scale-in then
        for each user $u_i$ in S do
            $N'_i \leftarrow$ number of requests for $u_i$;
            if $N' - N > M$ then
                Update trust value of user $u_i$ : $T_i = T_i \times \alpha$ ;
            end if
        end for
    end if
end for
```

Algorithm 5 Pseudo code for the MDAI trust value mitigation method

```
for each request r do
    if the trust value of request $T_i < T_{suspicious}$ then
        if $t_{forward}$ of $u_i > t_{release}$ then
            Update trust value of user $u_i$ : $T_i = T_i + \beta$ ;
        else
            Add user s to ALB forwarding rule;
        end if
        if the trust value of request $T_i$!‘$T_{malicious}$ then
            if $t_{block}$ of $u_i > t_{release}$ then
                Update trust value of user $u_i$ : $T_i = T_i + \beta$;
            else
                Add deny rule for $u_i$ in VPC ACL;
            end if
        end if
    end if
end for
```

5.1.5 Our Learning-Based Yo-Yo Mitigation Mechanism

In the preceding chapters, we established a theoretical framework for our Bayesian Stackelberg game theory model, aimed at tackling Yo-Yo attacks in cloud computing environments. Our model was meticulously designed to be applicable across a broad spectrum of cloud services. This chapter delineates the simulation experiments conducted to validate and refine our proposed model, making it not just versatile but also practically implementable.

Our provided mitigation solution commences with an intricate evaluation of data packets from users. This evaluation is bifurcated into two sequential phases: Classification and Attackers Type Recognition. Our defense mechanism begins by evaluating data packets received from each user. It involves a meticulous process of user classification and behavior analysis to dynamically label users as ”Normal,” ”Suspicious,” or ”Malicious.” This classification is crucial in identifying and mitigating potential Yo-Yo attacks.

The mechanism labels each user based on their interaction history and activity patterns. Newly added users or those with no prior suspicious activities are initially labeled as ”Normal.” The transition of a user's label from ”Normal” to ”Suspicious” and ultimately ”Malicious” hinges on the concept of critical Intervals. These intervals are defined as periods when the servers experience overload or are on the verge of triggering a scale-down process due to underload.

The attacker's type recognition phase is triggered only during every critical interval. At the end of each critical interval, whether leading to a scale change or not, our method initiates the SVM (Support Vector Machine) learning phase. Unlike the mentioned TASD methods, which update users' trust value only if that interval leads to a scale change. This phase analyzes user activity metrics, identifying clusters of users exhibiting similar behavior patterns. Users within these clusters are tagged as ”Suspicious,” yet allowed continued service access.

In subsequent critical intervals, if the SVM repeatedly identifies the same ”Suspicious” users exhibiting identical behavior, their label escalates to ”Malicious.” This transition leads to the blocking of their access and the dropping of all their subsequent requests.

This method only gives a one-time chance to a suspicious user before being dropped.

To refine the defense mechanism for mitigating Yo-Yo attacks and reduce the risk of false positives, adjustments can be made to provide more opportunities for users labeled as ”Suspicious” to demonstrate legitimate behavior. This can be achieved through two primary modifications: adjusting the trust value calculation and selectively blocking users based on their level of suspicion.

In order to revise our approach for managing ”suspicious” users to decrease the risk of false positives, we can utilize defining adjusting trust values, or we can use more customized selective blocking of suspicious users.

Adjusting Trust Values: For all users initially labeled as ”Normal,” assign a trust value of 1. After each critical interval, the cloud multiplies the trust value of these users by a discount factor within the range (0,1).

If a user’s trust value falls below a predefined threshold, their access is blocked. This approach allows users multiple chances to exhibit behavior before being labeled as ”Malicious.”

Selective Blocking of Suspicious Users: During critical intervals, the mechanism monitors the activities of all users tagged as ”Suspicious.” Instead of blocking all suspicious users simultaneously, the mechanism only blocks the top ’k’ most suspicious users.

The value of ’k’ is determined by the cloud operator, allowing for a more nuanced approach to blocking users and further reducing false positives.

Algorithm 6 shows the pseudo-code of our suggested mechanism for the detection and mitigation of Yo-Yo attacks.

5.1.6 Simulation Parameters

Our simulation environment is designed to closely mimic real-world cloud server scenarios. It features a cloud-based server that starts with a single instance, capable of handling 100 requests per millisecond. This setup initially caters to 10 active normal users, whose combined request load typically falls within the uncritical utilization range of 20-40%. To simulate attack scenarios, we introduce DDoS and Yo-Yo attacks of type 1, 2, and 3, each executed by 10 attacker bots. These bots are coordinated by an ’attacking brain’ which, crucially, has access to the server’s scaling state, mirroring sophisticated real-world cyber-attacks.

Algorithm 6 Pseudo code for our learning-based Yo-Yo mitigation mechanism

```
# Function to label a user
function LABELUSER(ip, activity, criticalInterval, svmModel)
    if ip not in registeredUsers then
        registeredUsers[ip] = ”label”: ”Normal”, “trustValue”: 1;
    end if
    if criticalInterval then
        if registeredUsers[IP][“label”] == “Normal” then
            # SVM learning phase;
            clusterLabel = svmModel.analyzeUserBehavior(activity)
            if clusterLabel == “Suspicious” then
                registeredUsers[ip][“label”] = “Suspicious”;
            end if
        else if registeredUsers[ip][“label”] == “Suspicious” then
            # Check for repeated suspicious behavior;
            if svmModel.confirmSuspiciousBehavior(activity) then
                registeredUsers[ip][“label”] = “Malicious”;
            end if
        end if
    end if
    # Update trust values and check for access block
    for userIp, userData in registeredUsers.items() do
        if userData[“label”] == “Normal” then
            userData[“trustValue”] *= discountFactor;;
            if userData[“trustValue”] ¡ threshold then
                registeredUsers[userIp][“label”] = “Malicious”;
            end if
        end if
    end for
end function
function PROCESSREQUEST(ip, activity, criticalInterval, svmModel)
    labelUser(ip, activity, criticalInterval, svmModel);
    userLabel = registeredUsers[ip][“label”];
    if userLabel == “Malicious” then
        return “Access Blocked”;
    else
        return “Access Granted”;
    end if
end function
```

The server's auto-scaling feature is configured to respond to fluctuations in demand. Under normal conditions, if server utilization exceeds 70% (equivalent to 70 requests per millisecond) for more than 20 seconds, it triggers an immediate scale-up, increasing the capacity from 1 instance to 4 instances. This expansion enhances the server's total handling capacity to 400 requests per millisecond. Conversely, after scaling up, if the server's utilization drops below 17.5% (70 requests per millisecond) for 20 consecutive seconds, it triggers a scale-down, reverting to its original single-instance configuration.

In our simulation, we streamline the process by minimizing the cool-down and warm-up intervals, which are not central to our key metrics. This means that adjustments in the number of instances due to auto-scaling occur instantaneously. New instances are added to the server as soon as they are launched, and similarly, during a scale-down, the extra instances are immediately removed without any delay. This approach allows us to focus on the core dynamics of the server's response to varying loads and attack patterns, providing us with clear insights into the efficacy of our defense mechanisms against different types of Yo-Yo attacks.

5.1.7 Designed Experiments

To rigorously evaluate the efficacy of our proposed mitigation solution against Yo-Yo attacks, we have conceptualized a series of simulation experiments. These experiments are meticulously designed to not only assess our solution but also to compare its performance with current state-of-the-art Trust-based Adversarial Scanner Delaying (TASD) methods: AD (Additive Decrease), ADAI (Additive Decrease/Additive Increase), and MDAI (Multiplicative Decrease/Additive Increase). The focus is on comprehensively analyzing the response of these systems to different types of Yo-Yo attacks, specifically types 1, 2, and 3.

In total, we have structured 8 simulation experiments, each tailored to provide insights into various aspects of the attack and defense dynamics. These experiments are carefully crafted to simulate real-world scenarios where cloud-based systems encounter diverse Yo-Yo attack patterns. Here are the key aspects these experiments aim to cover:

- *Attack Detection and Mitigation:* We will evaluate the ability of our solution to accurately detect and mitigate Yo-Yo attacks of different types. This includes assessing the speed and accuracy of attack detection, the effectiveness of response mechanisms, and the overall impact on system performance and stability.

- *Comparison with TASD Methods:* Each experiment is designed to provide a comparative analysis between our solution and the AD, ADAI, and MDAI TASD methods. This comparison will help us understand the relative strengths and weaknesses of our approach in various attack scenarios.
- *Types of Yo-Yo Attacks:* By focusing on Yo-Yo attacks of types 1, 2, and 3, we aim to cover a broad spectrum of auto-scaling aimed attack methodologies. This will allow us to evaluate the versatility and adaptability of our solution across different attack vectors.
- *Performance Metrics:* Key performance metrics such as total requests received from users, attackers' response to scaling fluctuations, trust-value adjustments, and the rate of instances utilizing false positives or negatives will be closely monitored. These metrics will provide quantitative data to support our analysis.
- *Real-World Scenario Simulation:* The experiments are designed to simulate real-world exploitation conditions as closely as possible, ensuring that our findings are applicable and relevant to actual cloud environments.

Through these comprehensive experiments, we aim to establish a clear understanding of the efficacy of our proposed solution in defending against different Yo-Yo attack types. The insights gained will not only validate our approach but also contribute to the ongoing efforts to enhance cloud security against such sophisticated cyber threats.

Experiment 1: DDoS Attack On an Auto-Scaling Enabled Server with No Defense Mechanism

The first experiment is implemented to assess the response of an auto-scaling server to a DDoS attack in the absence of any defense mechanisms. The experiment is set against a backdrop of regular traffic from normal users to mimic realistic conditions, with the primary goal of understanding the server's scaling behavior in reaction to the sudden surge of requests typical of a DDoS onslaught.

Experiment 2: Yo-Yo Attack Type 1 On an Auto-Scaling Enabled Server with No Defense Mechanism

The second experiment explores a simulation of a Type 1 Yo-Yo attack on a cloud service equipped with auto-scaling capabilities, conducted in the absence of any defense mechanisms. This approach aligns with the methodology outlined in reference [20], where the attacker employs probe requests to monitor and ascertain the service's scaling status, both for scale-up and scale-down phases.

Crucial to this experiment is the assumption that the attacker possesses a highly accurate understanding of the auto-scaling state, thereby maximizing the efficacy of the attack strategy. This level of insight is not solely reliant on data gathered by the attacker's bots. Instead, the attacker may acquire essential scaling information through external sources or by employing dummy users who consistently engage with the service in a normal manner. This approach ensures that any measures to delay or block specific attacker bots will not impede the attacker's access to vital information regarding the cloud service's scaling activities.

Experiment 3: Yo-Yo Attack Type 2 On an Auto-Scaling Enabled Server with No Defense Mechanism

In a manner akin to Experiment 2, this experiment involves simulating a Type 2 Yo-Yo attack on a cloud service equipped with auto-scaling functionality, conducted without the activation of any defense mechanisms.

Experiment 4: Yo-Yo Attack Type 3 On an Auto-Scaling Enabled Server with No Defense Mechanism

Following the methodology of Experiments 2 and 3, this phase of the study involves simulating a Type 3 Yo-Yo attack on a cloud service that is equipped with an auto-scaling feature, again conducted in the absence of any defense mechanisms.

Experiment 5: Test AD TASD Defense Mechanism With Different Yo-Yo Attack Types 1,2 and 3

In this experiment, the focus is on assessing the efficacy of the AD TASD mechanism in identifying and mitigating the three different types of Yo-Yo attackers. The primary objective is to evaluate how effectively the AD TASD approach detects and counteracts each specific Yo-Yo attack strategy.

The experiment adheres to parameters similar to those detailed in reference [20]. The initial trust value T_{init} is established at 10, a benchmark for evaluating user behaviors. To update this trust value, the parameter M is configured at 100, which influences the criteria for updating the trust value. Additionally, threshold values for classifying user activities are set, with $T_{suspicious}$ at 7 and $T_{malicious}$ at 5. These thresholds are instrumental in differentiating between normal, suspicious, and outright malicious activities based on the deviation of their behaviors from established norms. Furthermore, the parameter k, which determines the size of the list of users with the highest request load volume recorded by the detection module, is fixed at 10.

Experiment 6: Test ADAI TASD Defense Mechanism With Different Yo-Yo Attack Types 1,2 and 3

In this experiment, we assess the ADAI TASD defense mechanism's effectiveness against the three types of Yo-Yo attackers. The ADAI TASD, distinct from the AD TASD, places a greater emphasis on reducing false positives. It contemplates scenarios where a normal user's behavior temporarily appears suspicious, potentially leading to an incorrect decrease in their trust value and impacting service quality. To address this, ADAI TASD modifies the AD algorithm by managing the duration of request forwarding $t_{forward}$ or denial t_{block}, maintaining rules for a specific period $t_{release}$, set at 5 minutes, and then potentially increasing the trust value. Key parameters are similar to those in [20], with T_{init} set to 10, M at 100, and $T_{suspicious}$ and $T_{malicious}$ thresholds at 7 and 5, respectively.

Experiment 7: Test MDAI TASD Defense Mechanism With Different Yo-Yo Attack Types 1,2 and 3

In our analysis, we now examine the efficacy of the MDAI TASD defense mechanism in countering the three distinct types of Yo-Yo attackers. The MDAI TASD system adopts a similar approach to the ADAI method in prioritizing the reduction of false positives, but it employs a different set of metrics for trust value allocation. This method introduces a Multiplicative decrease strategy for trust value aimed at enhancing detection rates alongside an additive increase approach to minimize the occurrence of false-positive errors.

For this specific experiment, the initial trust value T_{init} is established at 100, with the variable M also set at 100 for updating the trust value. The multiplicative factor α is determined to be 0.5. This experiment also adjusts the thresholds for 'suspicious' $T_suspicious$ and 'malicious' $T_{malicious}$ statuses to 50 and 20, respectively. Additionally, the incremental factor β, which governs the rate of trust value increase, is set at 5, with this value incrementing every 1.5 minutes in accordance with the $t_{release}$ parameter.

Experiment 8: Test Our Presented Defense Mechanism With Different Yo-Yo Attack Types 1,2 and 3

Finally, it is time for us to evaluate the effectiveness of our provided learning-based defense mechanism against Yo-Yo attacks of types 1,2 and 3 and compare its efficacy to the AD, ADAI, and MDAI TASD mitigation mechanisms.

Distinct from the TASD mechanisms, which are reactive to scale changes (scale-up or scale-down) in server operations, our learning-based mechanism proactively updates its data inputs during any instances of overload or underload, regardless of whether these instances result in a scale change. This means that for TASD mechanisms to reassess users' trust values, a scale change must occur at the end of an overload or underload period. In contrast, our defense mechanism continuously updates its evaluations throughout these critical intervals, regardless of whether they culminate in a change in scale capacity or not.

In practical terms, our model operates as follows: When the cloud server experiences overload due to suspicious user activities, our mechanism captures user metrics at the end of this overload period, irrespective of a resulting scale change. It then employs an SVM classifier to identify patterns of behavior among users that could be contributing to these critical intervals. If the mechanism detects a group of users (e.g. π_i) exhibiting coordinated suspicious activities, it marks these users for further monitoring. Should these marked users repeat similar activities in subsequent critical intervals, the defense system intervenes by blocking the top k contributors within these groups, thus aiming to neutralize the threat.

Given the presence of ten attacker bots for each type of Yo-Yo attack, and in an effort to balance the reduction of false positives with high detection accuracy, we have configured the threshold k to be 5.

5.2 Experimental Results And Analysis

Eight distinct experiments were carried out to assess the effectiveness of various mitigation algorithms in response to different attack scenarios. This section delves into an analysis of the results obtained from these experiments, offering a detailed and comprehensive evaluation.

5.2.1 Results

Experiment 1

The first experiment illustrates the response of a cloud-based web service with auto-scaling capabilities to a single-phase Distributed Denial of Service (DDoS) attack. At the commencement of the simulation, typical users engage with the service, generating standard traffic. This is reflected in Figure 5.1, which depicts the aggregate requests per second from all active users.

Twenty seconds into the simulation, the orchestrator of the attack deploys DDoS tactics via bots, thereby escalating the total request rate. The strategy is to sustain an attack intensity that surpasses 70 packets per millisecond, triggering the service's scale-up threshold. Subsequently, 20 seconds post the onset of the DDoS attack, the cloud service's auto-scaling feature is activated, enhancing its processing capacity from 100 to 400 packets per millisecond. This adjustment in the cloud's processing capability, along with the concurrent total request rate, is shown in Figure 5.2.

One minute after the attack begins, the attacker ceases the assault, leading to a reduction in the request rate to below the 70-request threshold. A subsequent 20-second delay results in the triggering of the scale-down process, reverting the processing capacity from 400 back to 100 requests per millisecond.

Figure 5.3 presents the server's capacity Utilization chart, highlighting the number of requests relative to the server's maximum capacity at each point in time. The chart reveals a substantial increase in utilization, nearing full capacity, during the initial 20 seconds of the attack, prior to the activation of the scale-up threshold. Following the scale-up, utilization normalizes. Post-attack, there is a notable drop in utilization (below 10%), indicating significant excess capacity for the remaining normal traffic. Finally, 20 seconds after the cessation of the DDoS attack, the server's capacity is scaled down, and utilization levels return to their original state.

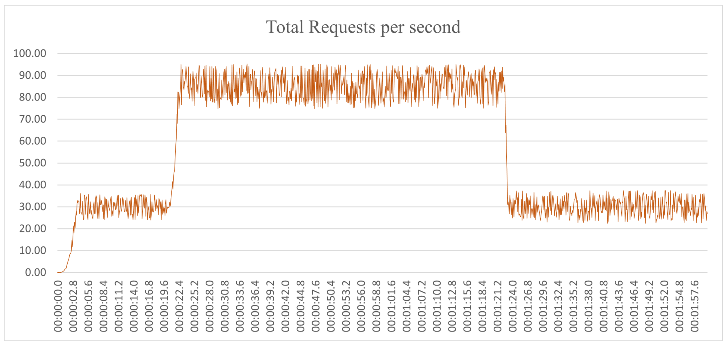


Figure 5.1: Total requests from all active users (normal users and attacker bots) per second to simulate a DDoS attack.

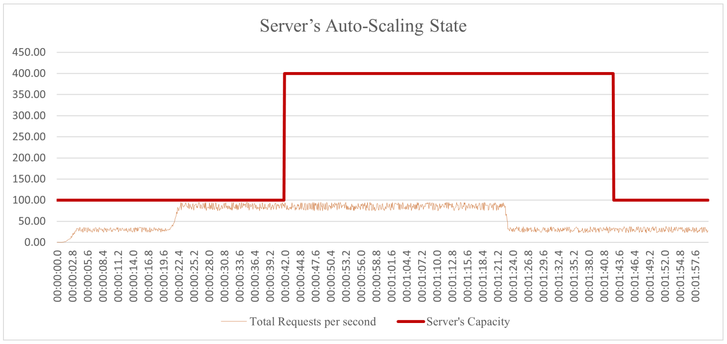


Figure 5.2: Cloud's processing capacity scale, together with the total requests per second

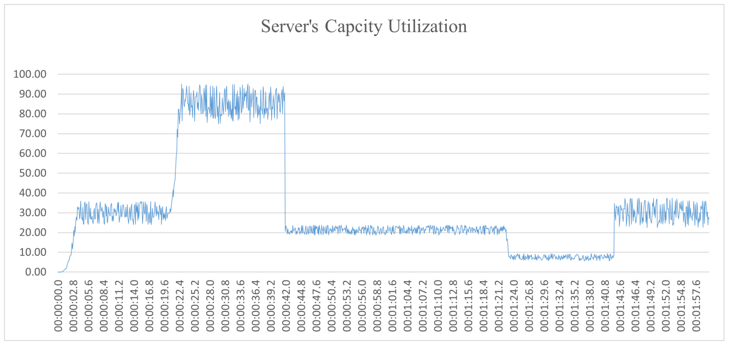


Figure 5.3: Server's Capacity Utilization

Experiment 2

The Type 1 Yo-Yo attack, as delineated in this experiment, follows a distinct tactical pattern aimed at exploiting the auto-scaling feature of the cloud service. The attack strategy involves intensifying the assault until the scale-up process is confirmed to be activated. Upon confirmation, the attack is abruptly halted, and the attacker then waits for the signal indicating the initiation of the scale-down process. As soon as this signal is detected, the same attack pattern is repeated, creating a continuous cycle of scaling up and down. This Type 1 strategy is recognized as the conventional Yo-Yo attack method in various related studies.

Figures 5.4 and 5.5 illustrate, respectively, the total requests per second and the scaling of the cloud's processing capacity. These visual representations clearly demonstrate that the Type 1 attack induces more significant fluctuations in the scaling state compared to Types 2 and 3. Furthermore, Figure 5.6, which showcases the utilization fluctuations, reveals a consistent pattern of oscillation. This pattern is characterized by alternating periods of high utilization (exceeding 75%) and very low utilization (below 10%).

Throughout the duration of this experiment, a total of 11 complete cycles of the Yo-Yo attack were observed, providing a comprehensive view of the impact and behavior of this type of Yo-Yo attacking strategy on cloud-based services.

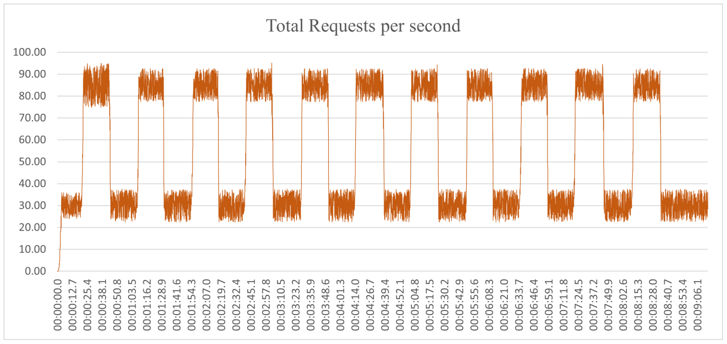


Figure 5.4: Total requests from all active users (normal users and attacker bots) per second to simulate a type 1 Yo-Yo attack

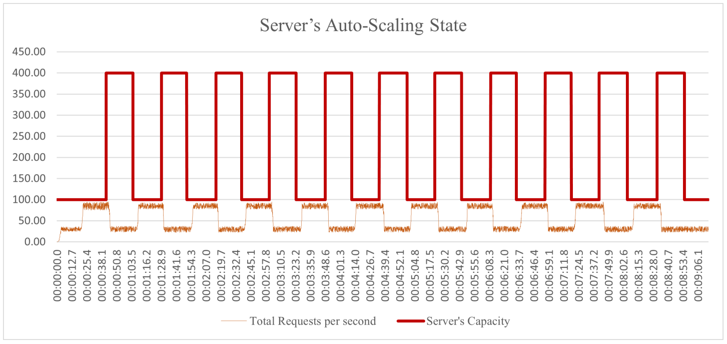


Figure 5.5: Cloud's processing capacity scale, together with the total requests per second

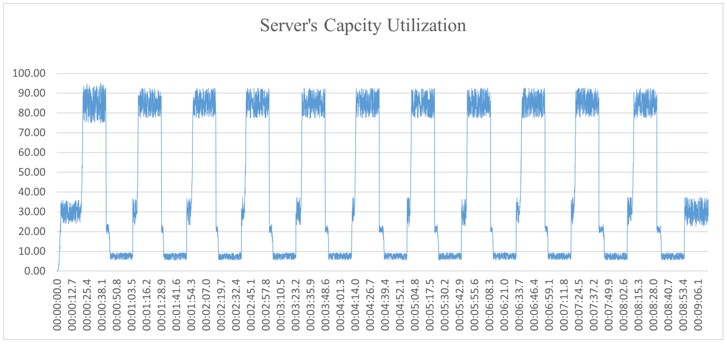


Figure 5.6: Server's Capacity Utilization

Experiment 3

While exhibiting cyclic attack patterns akin to those observed in Experiment 2, this experiment highlights a distinct operational focus for the Type 2 Yo-Yo attacker. The primary objective of this attacker is to maintain the servers at their maximum utilization capacity, strategically avoiding the activation of the scale-up mechanism.

Figure 5.7illustrates the fluctuations in the number of requests generated by the Type 2 attacker. These fluctuations are indicative of the attacker's efforts to modulate the intensity of the assault. The key distinction between the objectives of Type 1 and Type 2 attackers becomes evident in Figure 5.8. The Type 2 attacker, unlike its Type 1 counterpart, seeks to orchestrate periodic attacks that maximize server utilization without triggering a scale-up response.

Further evidence of this tactical approach is presented in Figure 5.9, which verifies the attacker's intention to sustain high utilization levels. However, it also demonstrates the attacker's deliberate reduction in attack intensity at specific intervals. This tactic is employed to prevent the engagement of the cloud service's 20-second scale-up threshold, thus maintaining a continuous high-load environment without prompting additional resource allocation.

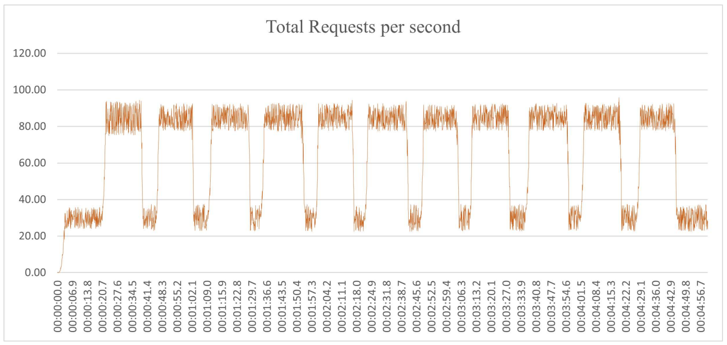


Figure 5.7: Total requests per second sent from all active users (normal users and attacker bots) to simulate a type 2 Yo-Yo attack

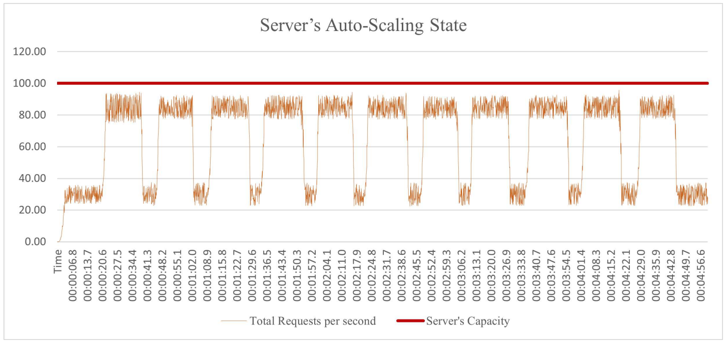


Figure 5.8: Cloud's processing capacity scale, together with the total requests per second

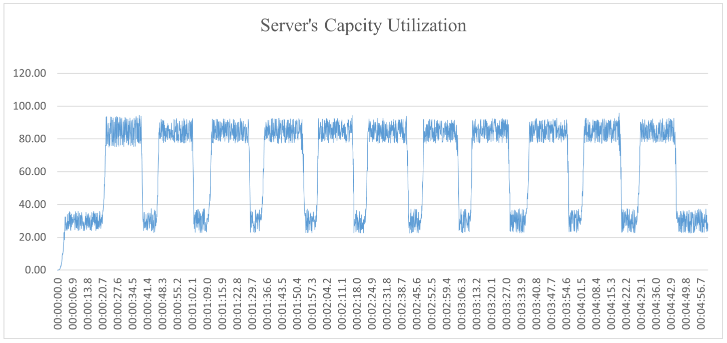


Figure 5.9: Server's Capacity Utilization

Experiment 4

In this experiment, we closely examine how the strategy employed by the Type 3 Yo-Yo attacker manipulates the performance statistics of the cloud server.

Figure 5.10 reveals that the initial wave of the Type 3 Yo-Yo attack is more prolonged than subsequent waves. This pattern becomes more apparent when analyzed in conjunction with Figures 5.11 and 5.12.

The Type 3 attacker commences with an extended first wave of attack, deliberately designed to ensure the triggering of a scale-up response from the server. Once the scale-up is confirmed to be in effect, the attacker shifts its approach. Instead of maintaining a constant high intensity of attack, it sporadically sends short bursts of data. The strategic intent behind these intermittent bursts is to disrupt and reset the scale-down triggering timer, effectively preventing the auto-scaling mechanism from initiating a scale-down process.

This tactic results in a scenario where the servers maintain and pay for additional capacity over an extended period, a capacity that remains primarily underutilized by legitimate, normal users.

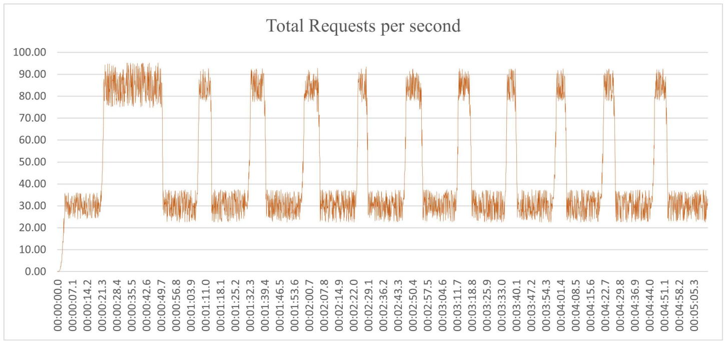


Figure 5.10: Total requests per second sent from all active users (normal users and attacker bots) to simulate a type 2 Yo-Yo attack

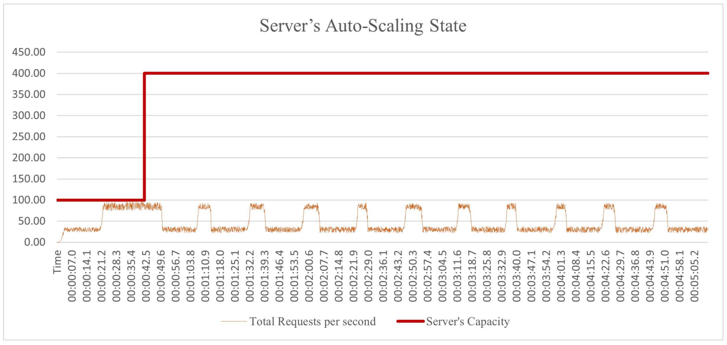


Figure 5.11: Cloud's processing capacity scale, together with the total requests per second

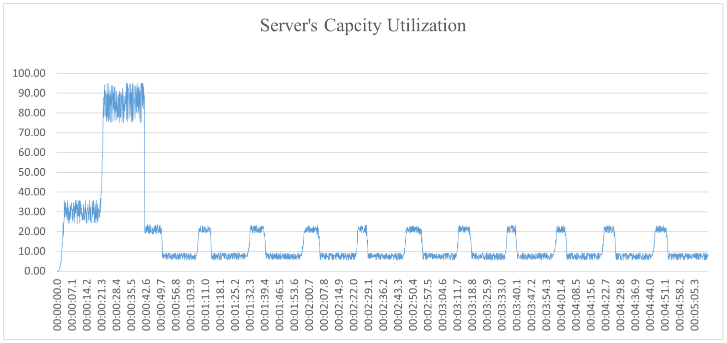


Figure 5.12: Server's Capacity Utilization

Experiment 5

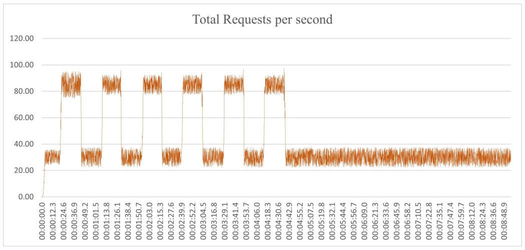
This experiment rigorously evaluates the effectiveness of the AD TASD mechanism in countering Yo-Yo attacks of types 1, 2, and 3.

The results for Type 1 attacks, as illustrated in Figure 5.13(a), indicate successful mitigation by the AD TASD mechanism, marking an improvement over Experiment 2, where no active defense was deployed.

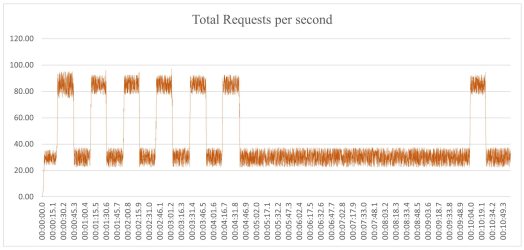
Conversely, Figures 5.15(a) and 5.17(a) reveal a stark contrast in the mechanism's performance against Type 2 and 3 attackers. Compared to Experiments 3 and 4, where no defense mechanisms were active, the AD TASD system demonstrates a notable deficiency in mitigating these types of attacks. The core limitation lies in the mechanism's reliance on scale changes to adjust users' trust values. Since Types 2 and 3 attacks do not oscillate the auto-scaling mechanism, they do not trigger changes in the attackers' trust values, thus evading detection. This limitation is further highlighted in Figures 5.16(a), 5.18(a), 5.19(b), and 5.19(c). The server utilization and scale fluctuations in these scenarios remain unaltered compared to the corresponding scenarios in Experiments 3 and 4, underscoring the ineffectiveness of the AD TASD mechanism against these attack types.

Additionally, the presumption that attackers have access to accurate scaling mechanism data undermines the effectiveness of delay injection strategies. Such attackers are not solely reliant on analyzing response times, thereby diminishing the impact of any delay tactics employed by the defense mechanism.

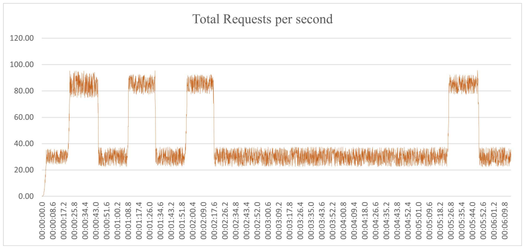
Interestingly, Figure 5.14(a) shows that while the Type 1 attacker initially succeeds in forcing six cycles of scale-ups, the defense mechanism eventually blocks their access once their trust value falls below the $T_{malicious}$ threshold. This leads to a cessation of their requests and a subsequent return of server utilization to normal levels, as depicted in Figure 5.19(a).



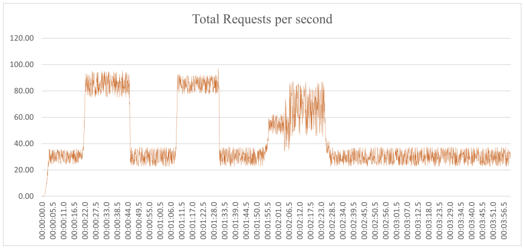
(a) AD TASD



(b) ADAI TASD

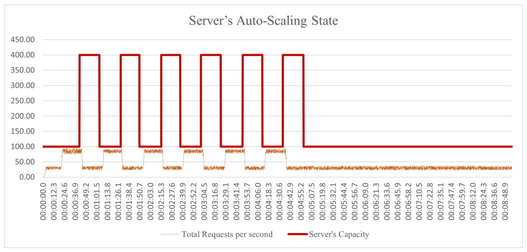


(c) MDAI TASD

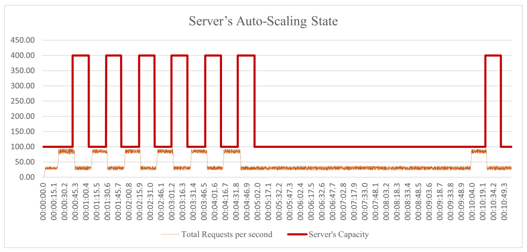


(d) Our defense mechanism

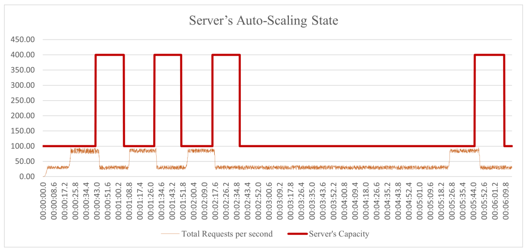
Figure 5.13: Total requests per second sent from all active users (normal users and attacker bots) to simulate a type 1 Yo-Yo attack on a server tested with different defense mechanisms.



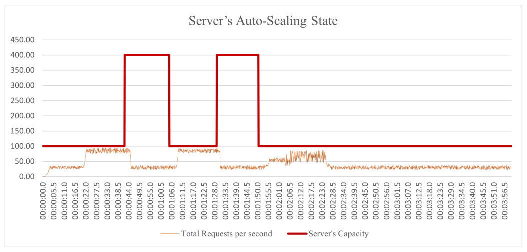
(a) AD TASD



(b) ADAI TASD



(c) MDAI TASD



(d) Our defense mechanism

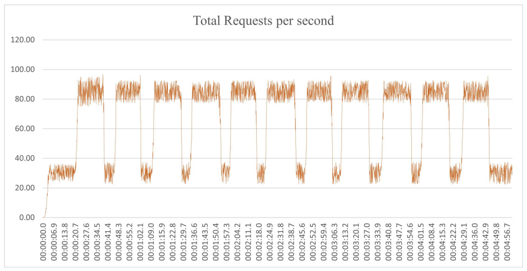
Figure 5.14: Cloud's processing capacity scale and the total requests per second in a Yo-Yo attack Type 1 against different defense mechanisms.

Experiment 6

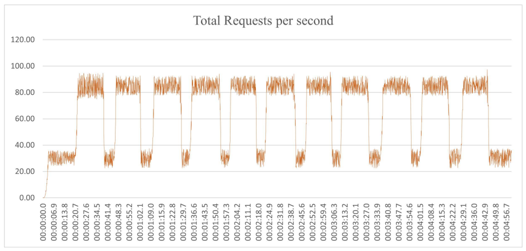
Similar to the previous experiment, the ADAI TASD methodology demonstrated effectiveness only in mitigating Type 1 attackers. This is illustrated in Figure 5.13(b), where the ADAI system successfully restricts attackers' server access following six scale oscillation cycles, paralleling findings from 5.13(a). A notable distinction in this experiment is the introduction of the $t_{release}$ parameter. This variable causes an incremental increase in the attackers' trust value by one unit, approximately five minutes subsequent to the initial access denial. Consequently, the trust value escalates from 4 to 5, altering the attacker's classification from 'malicious' to 'suspicious', thereby regaining server access. This cycle of the attacker resuming their attack pattern, as depicted in Figure 5.14(b), leads to another scale oscillation. The defense mechanism responds by reducing the trust value back to 4, reverting the attacker's status to 'malicious', resulting in the blockade of their requests. Figure 5.19(d) captures the server's utilization fluctuation induced by the Type 1 attack against the ADAI system. The pattern shows a potential perpetual oscillation between trust values of 4 and 5, given the recurrent pattern of trust value reduction and subsequent rise during the attack cycle.

Regarding Type 2 attackers, the ADAI system shows limitations in detecting their malicious activities as these attacks do not induce scale oscillations. This incapacity results in an unchanged trust value for these attackers, and consequently, the related data visualizations (Figures 5.15(b), 5.16(b), and 5.19(e)) resemble those from Experiment 3, where no defense mechanism was operational.

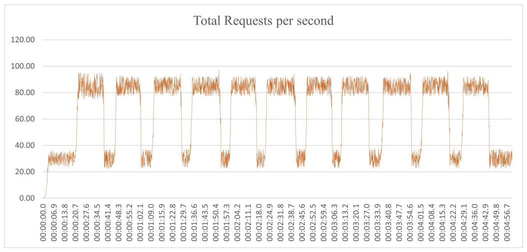
Similarly, the ADAI system fails to identify the activities of Type 3 attackers, as their strategy does not involve full scale-up and scale-down oscillations. This leads to a static trust value for Type 3 attacker bots, and as a result, the charts representing total requests count, autoscaling status, and utilization status (Figures 5.17(b), 5.18(b), and 5.19(f)) are comparable to those observed in Experiment 4.



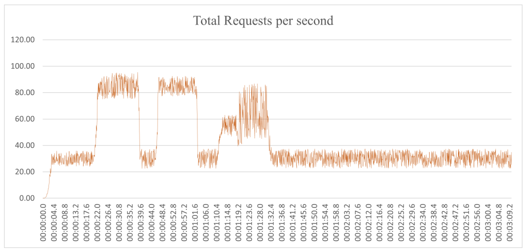
(a) AD TASD



(b) ADAI TASD

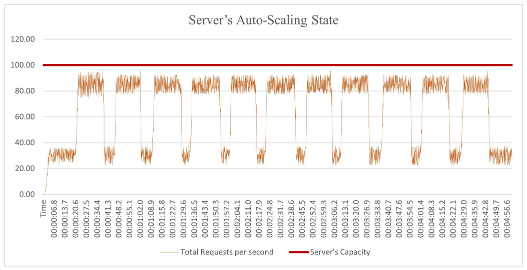


(c) MDAI TASD

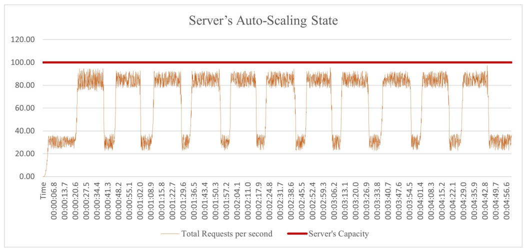


(d) Our defense mechanism

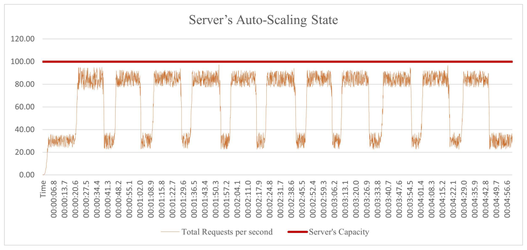
Figure 5.15: Total requests per second sent from all active users (normal users and attacker bots) to simulate a type 2 Yo-Yo attack on a server tested with different defense mechanisms.



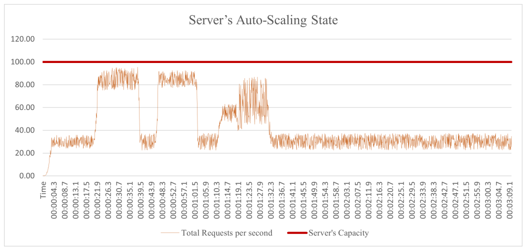
(a) AD TASD



(b) ADAI TASD



(c) MDAI TASD



(d) Our defense mechanism

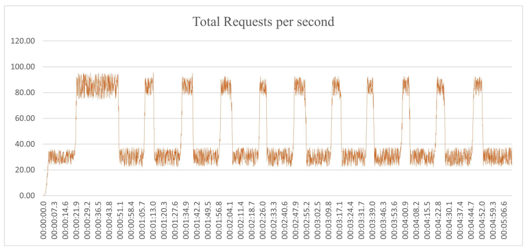
Figure 5.16: Cloud's processing capacity scale and the total requests per second in a Yo-Yo attack Type 2 against different defense mechanisms.

Experiment 7

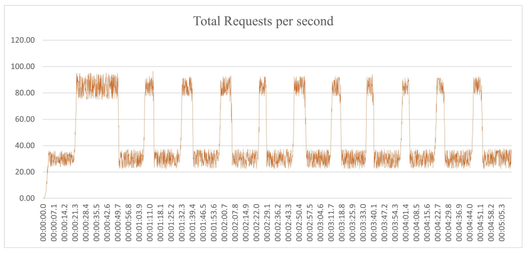
In this experimental evaluation, the MDAI TASD defense mechanism demonstrated effectiveness in mitigating only Type 1 Yo-Yo attacks but not Types 2 and 3.

As evidenced in Figure 5.13(c), the MDAI mechanism successfully neutralized the Type 1 attack within three oscillation cycles. This is a notable improvement in efficiency compared to the AD and ADAI methods, which required six cycles to mitigate the same type of attack. This enhanced performance is attributed to the MDAI's more aggressive approach in adjusting users' trust values. The trust value plummeted from 100 to 50 after the first oscillation, then further dropped to 25 after the second, and finally to 12.5 following the third attack wave. With the trust value falling below the threshold of 20, the attacker bots were effectively blocked by the server, initiating the $t_{release}$ countdown. Despite an incremental rise in trust value over two subsequent 1.5-minute periods and reaching over the 20 thresholds, the attackers managed to regain access, as shown in Figure 5.14(c). This pattern led to additional unnecessary scale oscillations, triggering the defense mechanism to reduce their trust value again, as illustrated in Figure 5.19(g), which depicts the utilization fluctuations caused by Type 1 attackers on the MDAI TASD-enabled cloud server.

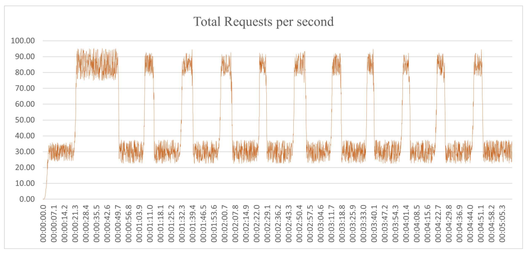
However, similar to the AD and ADAI TASD mechanisms, the MDAI method was unable to detect Type 2 and 3 attacks. For Type 2 attacks, the unchanged trust value resulted in outcomes parallel to those in Experiment 3, as shown in Figures 5.15(c), 5.16(c), and 5.19(h). Similarly, the MDAI mechanism could not effectively reduce the trust value for Type 3 attackers based on their malicious behavior. Consequently, the associated charts for Type 3 attacks, namely Figures 5.17(c), 5.18(c), and 5.19(i), exhibited no significant differences compared to those in Experiment 4. This indicates a consistent limitation in the TASD method's ability to detect and respond to these specific types of attacks.



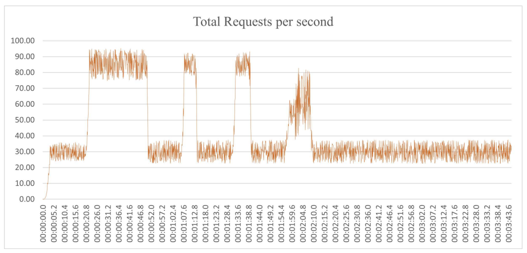
(a) AD TASD



(b) ADAI TASD

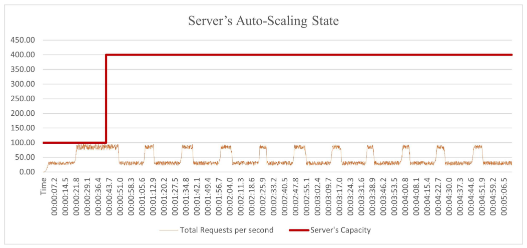


(c) MDAI TASD

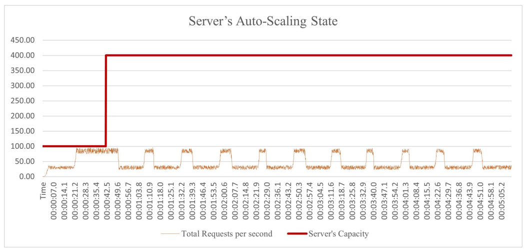


(d) Our defense mechanism

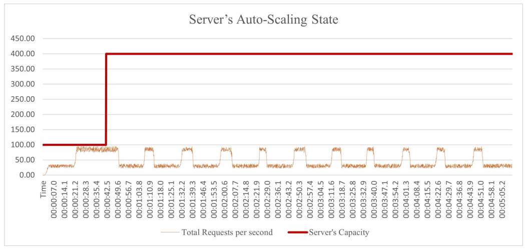
Figure 5.17: Total requests per second sent from all active users (normal users and attacker bots) to simulate a type 3 Yo-Yo attack on a server tested with different defense mechanisms.



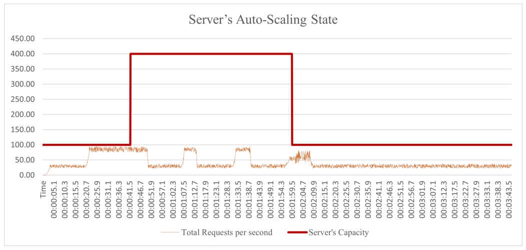
(a) AD TASD



(b) ADAI TASD



(c) MDAI TASD



(d) Our defense mechanism

Figure 5.18: Cloud's processing capacity scale and the total requests per second in a Yo-Yo attack Type 3 against different defense mechanisms.

Experiment 8

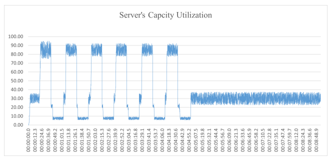
The comprehensive analysis of our learning-based defense mechanism against Yo-Yo attacks of Types 1, 2, and 3 demonstrates its superior efficacy compared to traditional AD, ADAI, and MDAI TASD mechanisms.

In the case of Yo-Yo attack Type 1, Figure 5.13(d) illustrates our defense mechanism's successful neutralization of this attack variant. Following the cessation of the initial attack wave, the system identified ten users exhibiting abnormal behavior during the first critical interval. At the conclusion of the second wave, the mechanism was re-evaluated to identify the group of users responsible for the subsequent system overload. The discovery that the same users instigated both waves led to the preemptive blocking of the top five attackers that caused the most disruption.

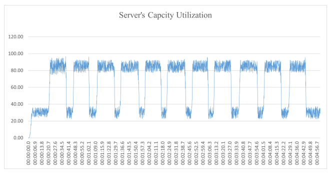
During the subsequent attack wave, the remaining five bots attempted to induce another scale oscillation, albeit with insufficient force. The attack orchestrator, recognizing this weakness, instructed the remaining bots to intensify their request rate. This strategy resulted in erratic efforts by the attackers, ultimately failing to prompt a further scale adjustment. The defense mechanism then identified and blocked these remaining bots. As depicted in Figures 5.14(d) and 5.19(j), the third wave's ineffectiveness in triggering a third scale-up and the subsequent normalization of server utilization underscore our mechanism's efficacy. Figures 5.13 and 5.14 further demonstrate the mechanism's superior performance in curtailing Type 1 attacks with minimal scale oscillations and preventing reentry of attackers, unlike ADAI and MDAI TASD mechanisms that inadvertently increased attackers' trust values.

Regarding Yo-Yo attack Type 2, Figure 5.15(d) shows our mechanism's effective mitigation comparable to that of Type 1. Despite Type 2's shorter attack waves, our system adeptly records and analyzes the attackers' patterns. Figures 5.16(d) and 5.19(k) depict the scale status during the simulation and the mechanism's restoration of normal server utilization after three attack waves. Figures 5.15 and 5.16 affirm our mechanism's unique capability in countering Type 2 attacks, which elude other TASD mechanisms due to the attackers' sophisticated strategies.

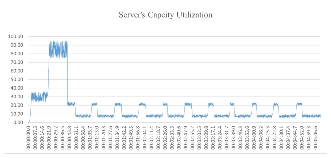
Finally, addressing Yo-Yo attack Type 3, Figure 5.17(d) confirms our defense mechanism's effectiveness in this scenario. Unlike the previous types, the mitigation of Type 3 required four waves, a variance explicated in Figure 5.18(d). The first wave's extended duration and scale impact, contrasted with the subsequent shorter waves aimed at preventing scale variation, presented a unique challenge. However, the defense mechanism adapted, identifying and mitigating five attackers by the third wave and the remainder by the fourth. Figure 5.19(l) illustrates the server utilization fluctuations throughout this process. Figures 5.17 and 5.18 further corroborate our mechanism's sole efficacy in mitigating Type 3 attacks, a feat unachievable by the other three TASD mechanisms.



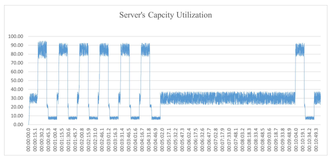
(a) Yo-Yo type 1 - AD TASD



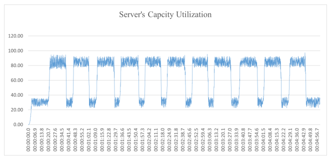
(b) Yo-Yo type 2 - AD TASD



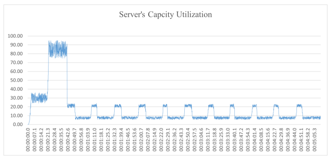
(c) Yo-Yo type 3 - AD TASD



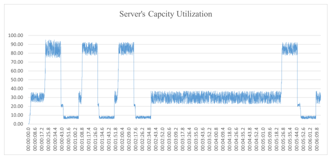
(d) Yo-Yo type 1 - ADAI TASD



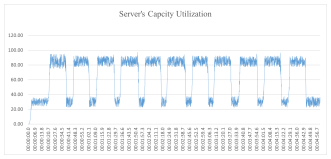
(e) Yo-Yo type 2 - ADAI TASD



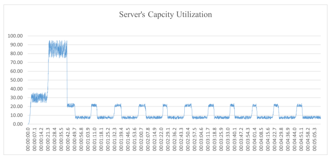
(f) Yo-Yo type 3 - ADAI TASD



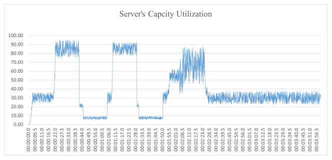
(g) Yo-Yo type 1 - MDAI TASD



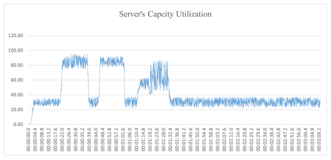
(h) Yo-Yo type 2 - MDAI TASD



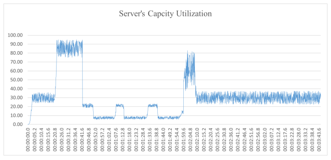
(i) Yo-Yo type 3 - MDAI TASD



(j) Yo-Yo type 1 - Our defense mechanism



(k) Yo-Yo type 2 - Our defense mechanism



(l) Yo-Yo type 3 - Our defense mechanism

Figure 5.19: Server's Capacity Utilization during Yo-Yo attack types 1, 2, and 3 tested with different defense mechanisms.

5.2.2 Discussion And Analysis Of The Experimental Results

Upon concluding all eight experiments and analyzing their outcomes, it becomes evident that our proposed defense mechanism outshines traditional TASD mechanisms in effectively countering various Yo-Yo attack types and in the promptness of attack mitigation.

This thesis introduces a sophisticated mitigation model based on a game-theoretical three-phase formulation, with learning-based type recognition as just one component. In more complex environments, the insights gained from the type recognition phase could be further incorporated into a MILP (Mixed Integer Linear Programming) optimization problem, enhancing the mechanism's robustness and adaptability.

Our experimental design was meticulously structured to facilitate a direct comparison between our advanced solution and the prevailing Yo-Yo TASD attack mitigation techniques. This comparative approach was integral to evaluating the efficiency of our mechanism relative to its predecessors.

Focusing on the Type 1 Yo-Yo attack, our analysis enables a comparison of different mechanisms' effectiveness. The AD TASD mechanism was neutralized after six attack waves, similar to the ADAI TASD. However, ADAI TASD's vulnerability, stemming from its false-positive avoidance policy, permitted attackers to reaccess servers after a designated $t_{release}$period. MDAI TASD exhibited improved performance over AD and ADAI, mitigating attacks in fewer waves, yet it shared the same flaw of readmitting attackers.

Our mechanism demonstrated superior performance, neutralizing attacks in the least number of waves, and crucially, unlike ADAI and MDAI, it permanently blocked malicious attackers from reaccessing the servers.

For Yo-Yo attacks of Types 2 and 3, the contrast becomes more pronounced as all TASD methods failed to mitigate attacks employing advanced strategies. As depicted in Figure 5.19, our mechanism uniquely restored server utilization to normal levels following all three attack types.

Chapter 6

Conclusions

This Thesis presents a comprehensive Repeated Dynamic Stackelberg Bayesian game model meticulously designed to determine the optimal strategy for safeguarding the cloud's auto-scaling mechanism. This model is structured in three pivotal phases: (1) The Dynamic Bayesian Stackelberg Game, which lays the foundational framework; (2) The Defense Module Mechanism, which actively counters threats; and (3) The Learning-Based Attackers' Type Recognition, which discerns and categorizes various attacker strategies.

A key contribution of this research lies in the detailed categorization of vulnerabilities in auto-scaling mechanisms, leading to the identification and analysis of more sophisticated exploitation strategies by attackers. This comprehensive analysis yielded the definition of three distinct types of auto-scaling exploitation Yo-Yo attack strategies, further enriching our understanding of the threat landscape. Significantly, our model demonstrates superior capability in mitigating various Yo-Yo attacking strategies. This is a noteworthy advancement over traditional TASD (Trust-based Adversarial Scanner Delaying) methods, which were limited to addressing a single type of periodic attack strategies. Our approach's versatility and adaptability mark a significant leap forward in cloud security protocols.

The robustness of the proposed solution was rigorously tested through cloud environment simulations, where it was benchmarked against contemporary Yo-Yo TASD attack mitigation methods, namely AD (Additive Decrease), ADAI (Additive Decrease/Additive Increase), and MDAI (Multiplicative Decrease/Additive Increase). The simulation results underscore the efficacy of our proposed solution.

A standout feature of our model is its marked improvement in detecting covert EDoS (Economic Denial of Sustainability) Exploitations. Compared to existing EDoS and Yo-Yo detection and defense strategies, our approach significantly reduces the percentage of successful attacks. This is a testament to the model's advanced predictive and reactive capabilities, positioning it as a formidable tool in cloud security.

As the digital landscape continually evolves, with threats becoming more sophisticated, our research offers a robust, efficient, and innovative solution. By setting a new benchmark in cloud security, this thesis addresses current security challenges and lays the groundwork for future advancements in the field. With its advanced approach, this model is poised to be a cornerstone in the ongoing effort to secure cloud environments against increasingly sophisticated threats.

Bibliography

[1] D. MacRae, “81% of firms have accelerated their cloud computing plans due to COVID-19.” https://www.cloudcomputing-news.net/news/2021/jun/14/81-of-firms-have-accelerated-their-cloud-computing-plans-due-to-covid-19/, June 2021.

[2] M. G. Avram, “Advantages and Challenges of Adopting Cloud Computing from an Enterprise Perspective,” *Procedia Technology*, vol. 12, pp. 529–534, Jan. 2014.

[3] R. Ravichandiran, H. Bannazadeh, and A. Leon-Garcia, “Anomaly Detection using Resource Behaviour Analysis for Autoscaling systems,” in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pp. 192–196, June 2018.

[4] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, “Inferring Internet denial-of-service activity,” *ACM Trans. Comput. Syst.*, vol. 24, pp. 115–139, May 2006.

[5] B. B. Gupta and O. P. Badve, “Taxonomy of DoS and DDoS attacks and desirable defense mechanism in a Cloud computing environment,” *Neural Comput & Applic*, vol. 28, pp. 3655–3682, Dec. 2017.

[6] M. Darwish, A. Ouda, and L. F. Capretz, “Cloud-based DDoS attacks and defenses,” in *International Conference on Information Society (i-Society 2013)*, pp. 67–71, June 2013.

[7] A. Bello Usman and J. Gutierrez, “Toward trust based protocols in a pervasive and mobile computing environment: A survey,” *Ad Hoc Networks*, vol. 81, pp. 143–159, Dec. 2018.

[8] K. Singh, P. Singh, and K. Kumar, “Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges,” *Computers & Security*, vol. 65, pp. 344–372, Mar. 2017.

[9] A. Bonguet and M. Bellaiche, “A Survey of Denial-of-Service and Distributed Denial of Service Attacks and Defenses in Cloud Computing,” *Future Internet*, vol. 9, p. 43, Sept. 2017.

[10] G. Sun, V. Chang, M. Ramachandran, Z. Sun, G. Li, H. Yu, and D. Liao, “Efficient location privacy algorithm for Internet of Things (IoT) services and applications,” *Journal of Network and Computer Applications*, vol. 89, pp. 3–13, July 2017.

[11] M. A. S. Monge, J. M. Vidal, and L. J. G. Villalba, “Entropy-Based Economic Denial of Sustainability Detection,” *Entropy*, vol. 19, p. 649, Dec. 2017.

[12] M. Sides, A. Bremler-Barr, and E. Rosensweig, “Yo-Yo Attack: Vulnerability In Auto-scaling Mechanism,” *SIGCOMM Comput. Commun. Rev.*, vol. 45, pp. 103–104, Sept. 2015.

[13] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, “Split/Merge: System Support for Elastic Execution in Virtual Middleboxes,”

[14] Z. A. Baig and F. Binbeshr, “Controlled Virtual Resource Access to Mitigate Economic Denial of Sustainability (EDoS) Attacks against Cloud Infrastructures,” in *2013 International Conference on Cloud Computing and Big Data*, pp. 346–353, Dec. 2013.

[15] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang, “Reduction of Quality (RoQ) Attacks on Dynamic Load Balancers: Vulnerability Assessment and Design Tradeoffs,” in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, pp. 857–865, May 2007.

[16] “Are You Protected Against Burst Attacks? – Radware Blog.” https://www.radware.com/blog/security/2018/02/burst-attack-protection/.

[17] A. Bremler-Barr, E. Brosh, and M. Sides, “DDoS attack on cloud auto-scaling mechanisms,” in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9, May 2017.

[18] “AWS Best Practices for DDoS Resiliency - AWS Best Practices for DDoS Resiliency.” https://docs.aws.amazon.com/whitepapers/latest/aws-best-practices-ddos-resiliency/aws-best-practices-ddos-resiliency.html.

[19] X. Xu, J. Li, H. Yu, L. Luo, X. Wei, and G. Sun, “Towards Yo-Yo attack mitigation in cloud auto-scaling mechanism,” *Digital Communications and Networks*, vol. 6, pp. 369–376, Aug. 2020.

[20] M. M. Kashi, A. Yazidi, and H. Haugerud, “Mitigating Yo-Yo attacks on cloud auto-scaling,” in *2022 14th IFIP Wireless and Mobile Networking Conference (WMNC)*, pp. 46–53, Oct. 2022.

[21] T. Lorido-Botrán, J. Miguel-Alonso, and J. Lozano, “A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments,” *Journal of Grid Computing*, vol. 12, Dec. 2014.

[22] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, “Resource-Aware Detection and Defense System against Multi-Type Attacks in the Cloud: Repeated Bayesian Stackelberg Game,” *IEEE Transactions on Dependable and Secure Computing*, vol. 18, pp. 605–622, Mar. 2021.

[23] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, “Optimal Load Distribution for the Detection of VM-Based DDoS Attacks in the Cloud,” *IEEE Transactions on Services Computing*, vol. 13, pp. 114–129, Jan. 2020.

[24] M. Ranganath and M. Keating, “How to detect suspicious activity in your AWS account by using private decoy resources — AWS Security Blog.” https://aws.amazon.com/blogs/security/how-to-detect-suspicious-activity-in-your-aws-account-by-using-private-decoy-resources/, Aug. 2022.

[25] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[26] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, “I Know You Are Watching Me: Stackelberg-Based Adaptive Intrusion Detection Strategy for Insider Attacks in the Cloud,” in *2017 IEEE International Conference on Web Services (ICWS)*, pp. 728–735, June 2017.

[27] A. Hota, A. Clements, S. Sundaram, and S. Bagchi, "Optimal and Game-Theoretic Deployment of Security Investments in Interdependent Assets," pp. 101–113, Nov. 2016.

[28] A. Clark, K. Sun, L. Bushnell, and R. Poovendran, "A Game-Theoretic Approach to IP Address Randomization in Decoy-Based Cyber Defense," pp. 3–21, Nov. 2015.

[29] M. Irvine, "Average Cost per Click by Country [DATA]." https://www.wordstream.com/blog/average-cost-per-click.

[30] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus, "Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '08, (Richland, SC), pp. 895–902, International Foundation for Autonomous Agents and Multiagent Systems, May 2008.

[31] M. R. Watson, N.-u.-h. Shirazi, A. K. Marnerides, A. Mauthe, and D. Hutchison, "Malware Detection in Cloud Computing Infrastructures," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, pp. 192–205, Mar. 2016.

[32] "Change the desired capacity of an existing Auto Scaling group - Amazon EC2 Auto Scaling." https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-manual-scaling.html.

[33] "Dynamic scaling for Amazon EC2 Auto Scaling - Amazon EC2 Auto Scaling." https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html.