Thimblerig: Architectural Moving Target Defense and a framework for its Game theoretic analysis

Gautam Kumar

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science in

Computer Science and Software Engineering

University of Washington

2017

Committee:

Brent Lagesse

Kelvin Sung

Munehiro Fukuda

James Fridley

Program Authorized to Offer Degree:
Computing and Software Systems

University of Washington

Abstract

Thimblerig: Architectural Moving Target Defense and a framework for its Game theoretic analysis

Gautam Kumar

Chair of the Supervisory Committee:

Enterprises today are rapidly moving their internal server infrastructure to cloud providers such as AWS and Azure. Migrating to IaaS providers raises a different set of security threats when compared to in-house IT infrastructure. But cloud infrastructure also offers new sets of opportunities and flexibility which were previously infeasible. One such opportunity is implementing a Moving Target Defense (MTD) as part of a cloud deployment's architecture. In this thesis we proposed a MTD architecture inspired by Netflix's chaos monkey library. We evaluated our proposed architecture for performance overhead, ran discrete event simulations to evaluate potential benefits in terms of cost of being exploited and finally we model our architecture as a game to formulate a function to compute the probability of attack. Our goal with this work was to highlight the possibility of implementing a layer of moving target defense at an architectural level for enterprise security teams

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

According to Mcafee [2], cloud services are now an integral component of IT operations in more than 90% of organizations in the world. The highest growth in cloud services for the year 2017 is expected to be in the Infrastructure as a Service (IaaS) sector. IaaS services are expected to grow at nearly 36% to reach a world wide spending of about $34 billion dollars [6]. With such rapid adoption of cloud services, cloud deployments are becoming an worthy target for malicious actors [12].

To safeguard cloud deployments organizations are turning towards the classic multi-layered security solutions [59, 43] which are common in physical security scenarios. One potential layer of security is Moving Target Defense (MTD). The Department of Homeland Security defines MTD as the concept of controlling changing across multiple system dimensions in order to increase uncertainty and apparent complexity for attackers, reduce their window opportunity and increase the cost of their attack efforts [5].

A lot of MTD research today is done at the network level [18, 38, 29]. Network based MTD is still extremely relevant today, especially with the advent of SDNs being deployed in cloud infrastructure. We decided to stray from the pack and direct our efforts at building a MTD based architecture for the cloud. So in this thesis we propose a MTD architecture for the cloud, with the goal of significantly increasing the cost for an attacker. Our proposed architecture introduces MTD in the form of ephemeral servers with a limited lifetime (TTL). The TTL of each server is based on factors which are difficult to predict by an attacker. We then proceed to simulate our architecture to quantify the potential benefits that a defender may gain along with

the increase in costs to an attacker. The primary audience for our work are security teams within enterprises.

Our work in this architecture was inspired by Netflix's chaos monkey, a tool designed to test reliability of cloud infrastructure deployments (see section 3.1). We postulated that we could leverage the uncertainty introduced by a tool such as chaos monkey to improve security within an organization and we test this belief in chapter 5.

Chapter 2

BACKGROUND

2.1 State of cloud infrastructure

Cloud computing is an umbrella term which refers to the usage of shared compute resources on demand. In simpler terms cloud computing could be considered analogous to renting computational resources as and when its needed. There are three primary layers of cloud computing, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (Software as a Service). Our work in this thesis applies only at the IaaS layer. Figure 2.1 visually illustrates the layers of cloud computing.

IaaS services such as Amazon's AWS, Microsoft's Azure, and Google's Cloud Platform offer a myriad of products as part of their IaaS line up with Compute, Storage and Database being the most widely used products [11].

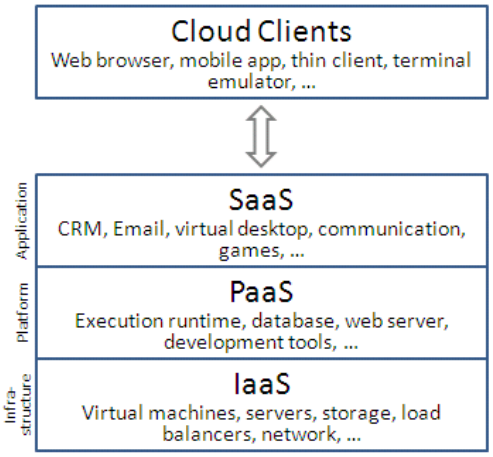

Figure 2.1: Cloud Computing Layers

2.2 Cryptography

2.2.1 Cryptographic hash function

A cryptographic hash function is any one way function which meets the following requirements [46]

- Preimage resistance

- Collision resistance
- Second Preimage resistance

A hash function has preimage resistance if given a hash value h it is computationally infeasible to find any message m such that $h = hash(k, m)$ where k is the hash key.

A hash function is collision resistant if, given two messages m_1 and m_2 it is hard to find a hash h such that $h = hash(k, m_1) = hash(k, m_2)$ where k is the hash key.

A hash function has second pre-image resistance if given a message m_1 it is computationally infeasible to find a different message m_2 such that $hash(k, m_1) = hash(k, m_2)$ where k is the hash key. The second pre-image resistance is a much harder property to achieve for hash functions. This property is closely related to the birthday problem [36].

2.2.2 Hash Chains

Leslie Lamport [35] first proposed the use of hash chains in his paper on a method for secure password authentication over an insecure medium. One of the properties of hash chain-based authentication is that it has a limited use lifespan. We leverage this property in our system to limit the damage

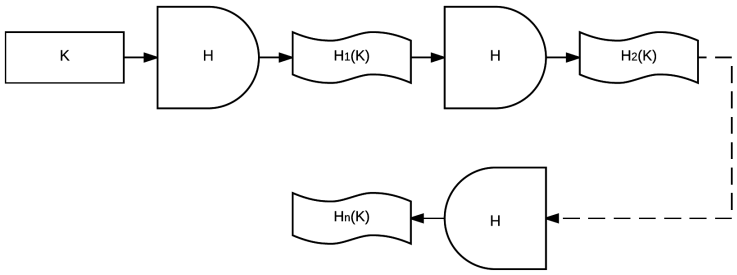

Figure 2.2: A simplified view of a hash chain where the start message is K with H being the cryptographic hash function. $H_1(K)$, $H_2(K)$ and $H_n(K)$ represent successive messages in the Hash chain

that any single compromised machine can cause.

"A hash chain is a sequence of values derived via consecutive applications of a cryptographic hash function to an initial input. Due to the properties of the hash function, it is relatively easy to calculate successive values in the chain but given a particular value,it is infeasible to determine the previous value"

A hash chain in essence is merely the successive computation of a Cryptographic hash function on a given value as demonstrated in figure 2.2.

As an example, Let x be the initial password and H be the cryptographic hash function. A hash chain of length 2 would be $H^2(x) = H(H(x))$. A hash chain of n values is denoted as $H^n(x)$ and the i^{th} value in the chain would be computed as $x_i = H(x_{i-1})$.

For a given value in the chain x_i its computationally infeasible to determine the previous value in the chain x_{i-1}.

2.3 Security in the cloud

Securing cloud systems is a multi-specialty field. Cloud infrastructure providers such as Amazon consider security from multiple different perspectives. As an example, AWS security is implemented as a multi layered solution ranging from physical security to hypervisor and network security [1]. While these measures are essential for maintaining the integrity, reliability, and security of cloud infrastructure, they are also purpose built to secure only Amazon's (or any cloud provider) infrastructure and the service they offer customers.

Amazon highlight the importance of share responsibility in their cloud security information page [9] with a diagram (fig. 2.3) which emphasizes the importance of AWS customers implementing security measures to resources within their purview.

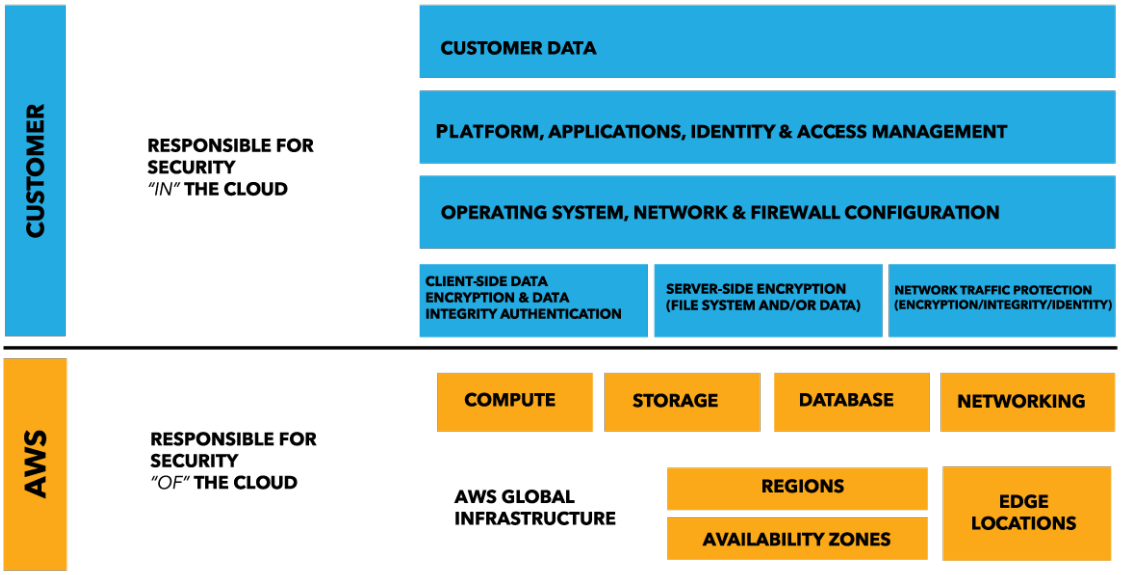

Figure 2.3: Amazon's shared responsibility model for cloud security. Source [9]

2.4 Moving target defense

Moving Target Defense (MTD) is a defensive strategy which aims to increase the uncertainty, complexity, and cost for attackers with relatively low impedance to the defender [5]. The aim of MTD is not attack prevention, rather, the aim of MTD is to increase the cost and decrease the value of attacking a system.

2.5 Game Theory

The goal of game theoretic analysis is to model the interactions between two or more players of a game. A game in this scenario refers to a situation where the outcome is determined based on the strategies of each player. A player in this scenario is any actor who's actions can influence the outcome of a game. Each player has a limited set of actions which are known as strategies.

Scenarios are modeled as games to simplify the process of reasoning through complex and unintuitive situations. When modeling a game each player has a set of

possible strategies and their respective pay-offs. A pay-off is the value that a players derives as a consequence of the outcome of the game.

Strategies could potentially have a cost associated with it and based on the outcome of the game each strategy could have a benefit as well. The total value that a player derives from this is known as the utility. Thus the utility of a strategy can be defined as $Utility = Benefit - Cost$.

2.5.1 Nash equilibrium

A classic example of a game is the prisoner's dilemma. This game consists of two players who can each have two strategies, Co-operate or Defect. If both players co-operate they both get a pay-off or 2. If one player defects while the other co-operates the player who defects gets a pay-off 3 while the player who co-operated gets no pay-off. Finally if both players defect, they each get a pay-off of one. The pay-off matrix for the classic prisoner's dilemma game is shown in table 2.1

Table 2.1: 2x2 Matrix: Prisoner's Dilemma

		Player 2	
		C	D
Player 1	C	$2, 2$	$0, 3$
	D	$3, 0$	$1, 1$

The Prisoner's dilemma game clearly illustrates the concept of Nash Equilibrium. Nash equilibrium is the state of a game in which no player can gain any utility by changing their current strategy. The Nash Equilibrium within the prisoner's dilemma game is for both players to defect. This is counter intuitive as co-operating yields a better pay-off for both players. Since each player has no way of learning the preference of the other player it is in each rational player's best interest to defect.

2.5.2 Pure and Mixed Strategies

A mixed strategy is a strategy which would be played with a particular probability. For example, if we know that an un-fair coin lands as heads 45% of the time and as tails 55% of time, then a player should ideally play heads with a probability of 0.45 and tails with a probability of 0.55. A pure strategy is considered to be mixed strategy with a probability of 1.0.

2.5.3 Mixed Strategy Nash equilibrium

When simulating real life scenarios we encounter games which don't have a simple pure strategy Nash Equilibrium as seen in the classic Prisoner's dilemma game. But Nash's theorem states that there is at-least one equilibrium for all finite games [42]. A finite game being a game with a finite set of players each with a finite set of strategies. So if a equilibrium does not exist in pure strategies, then one must exist in mixed strategies.

We can compute the mixed strategy Nash Equilibrium by setting the expected payoff function of each action that a player could take equal to the expected payoff function of an alternative action. By solving these set of equations we can obtain the mixed strategy Nash equilibrium for each player.

Chapter 3

RELATED WORK

In this chapter we provide a brief overview of the current research landscape surrounding Moving Target Defense, Cloud Security and relevant Game Theory. We also highlight papers which are relevant, or have inspired parts of this thesis.

3.1 Chaos Monkey

Chaos monkey [52, 14] is a tool developed by Netflix. The purpose of this tool is to continuously test reliability of their cloud infrastructure which is hosted on AWS [4]. Chaos monkey tested reliability by randomly selecting virtual machine instances within netflix's production infrastructure pool and terminated them. Terminating a virtual machine instance is equivalent to shutting down a computer and trashing its components. This means that the terminated instances are not recoverable. The primary reason for building such a destructive tool was to encourage engineers to design and build software services which are resilient unpredictable failure.

Chaos monkey served as an inspiration for our work in this thesis. Our proposed architecture aims to leverage the unpredictable nature of failure events in a controlled manner and apply that principle to securing cloud infrastructure using moving target defense.

3.2 Cloud security

Research today into cloud security, especially within the scope of IaaS providers, is focused on two primary areas, Securing VM infrastructure [51, 49, 40, 20] and providing tools for implementing security at the customer level (as described in section 2.3).

Our goal with this thesis is to provide an additional tool in the form of a MTD Architecture which can be implemented at the customer level.

3.3 Current research in MTD

A lot of research has been done in moving target defenses. Much of the primary focus in MTD research has been on implementing MTD at various levels in a network. For example Dunlop et al.[18, 23, 56, 55] developed MTD6, a system for leveraging the vast address space of IPv6 to improve user privacy and protect against targeted network attacks. Cloud infrastructures use virtualized networks which are commonly referred to as Software defined networks (SDN). In SDN based MTD research [38, 29] the authors attempt to leverage the highly flexible nature of software defined networks to obfuscate the attack surface and decrease an attacker's ability to identify targets.

Green et al. [22] identify seven different criteria for network based moving target defenses. These criteria are, Unpredictability, Vastness, Periodicity, Uniqueness, Availability, and Revocability. In section 6.1.2 we try define how our implementation conforms to some of these criteria.

A less related but important area of MTD research focuses on implementing MTD within compilers to decrease an attacker's ability to exploit memory corruption vulnerabilities. A classic example of this is ASLR and code diversification [30, 50, 26, 27]. Address Space Layout Randomization (ASLR), as the name suggests, randomly arranges the locations of important data parts of a process, such as stack, heap, libraries and the executable itself. Such a random arrangement increases unpredictability and acts as a deterrent to memory corruption vulnerabilities.

3.4 Game theory and Security

As cyber-security research has gained much more prominence over the last decade, a lot of researchers have examined cyber security from a game theoretic perspective [39, 60, 28, 31, 48]. An example of such research is the work of Fan et al. [19]. The

authors of this paper model the interactions between defenders and attackers as a Stochastic game. A stochastic game is with multiple stages and one or more players. Each stage transition has a probability. By modeling cloud computing architectures as a stochastic game the authors are able to dynamically evaluate multiple defense strategies to discover the best one based on the current scenario. The authors of the paper describe evaluating 15 different strategies dynamically at runtime.

Similarly, Furuncu & Sogukpinar [21] analyze the security in IaaS cloud deployments using a normal form game to evaluate the costs and benefit that attackers and defenders encounter. Based on their analysis the authors claim that if an attacker attacks more than 76% of systems within an organization, then the cost of taking security measures would out weigh the benefits.

Our work in section 5.6 of adapting to a Nash Equilibrium attacker is inspired by the work done on game theory based security mechanism for mobile P2P systems [33, 34]. The authors propose a game theoretic model for determining the pay-offs for an attacker and defender. Using these pay-offs the authors determine the mixed strategy Nash Equilibrium and the probability of attack.

3.5 A study on time to compromise

Hannes Holm, a researcher in Sweden, conducted a large scale study[24] for 434 days to determine the mean Time to Compromise (TTC), Time to First Compromise (TTFC) and Time between Compromises (TBC) of computer systems. He discovered the mean TTFC to be 171 days, the mean TBC to be 75 days and the overall TTC as 108 days. We use facts presented in this work as the basis for the simulations we describe in chapter 5. For example we use 10416 as the maximum simulation time as 10416 is the number of hours in 434 days.

3.6 FlipIt

The concept of using ephemeral servers as a mechanism for moving target defense is referenced by Dijk et. al. [53]. The authors propose a two player game where each player competes to maximize the amount of time that they have access to a resource while minimizing their costs. The unique aspect of their game design is the stealth aspect. The stealth aspect is the concept that a player can learn about the current state of the system only when they move. Each move by a player has an associated cost. A player's benefit is determined by the amount of time that they posses control of the system. The authors go on to describe multiple different strategies for various scenarios. This paper highlights a unique direction for analyzing our architecture using game theory and suggests a possible route for future work.

Chapter 4

METHODS

4.1 Architecture

Our proposed architecture consists of two types of servers, Client Facing Servers (CFS) and a Central Trusted Authority (CTA). The CFS and CTA are classifications which refer to two commonly used server types. A CFS is any server which is capable of communicating outside of a virtual cloud network, while a CTA is any server which is inherently responsible for securing a sensitive resource. Finally a sensitive resource refers to any asset or service which needs to be protected from unauthorized access. Examples of sensitive resources include data stores such as databases, and authentication information to external services such as API Keys. A high level overview of this architecture is illustrated by fig. 4.1.

4.1.1 Enforcing moving target defense through limited use

One of the core components of our proposed architecture is the concept of limited use tokens which enforce moving target defense. We decided to use Hash Chains as a

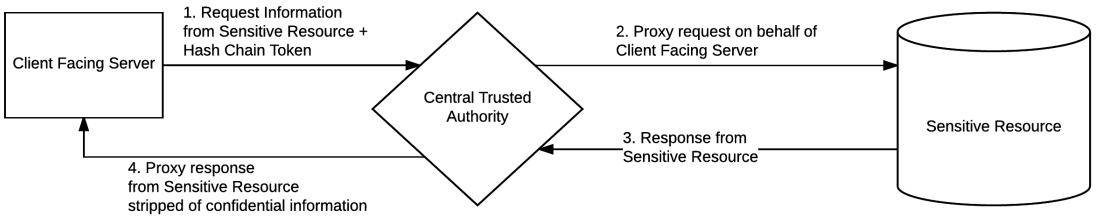

Figure 4.1: A high level overview of our proposed architecture

mechanism to implement limited use tokens. Hash Chains inherently offer guarantees of limited use as described in section 2.2.2.

Timers, or counters can be used in place of a hash chain as they can be used to enforce moving target defense through limited use. A timer or counter setup is relatively simple to implement when compared to a Hash Chain mechanism, but timers and counters lack the cryptographic guarantee that a hash chain provides. The lack of cryptographic guarantees increases the risk of spoofing by an attacker to significantly extend the lifetime of a CFS. The ability to dynamically increase the lifetime of a server defeats the purpose of our architecture. Thus we chose to as a mechanism for enforcing limited use.

4.1.2 Central Trusted Authority

The Central Trusted Authority consists of three primary components, A hash chain verifier, A storage back-end and a request proxy. The CTA performs three roles within the system, which are

- Create new hash chains
- Verify hash chains
- Proxy requests to resources

Creating a new hash chain: Hash chains are created by iteratively hashing a secret token T, n number of times. After the hash chain is created the CTA stores $H^n(T)$ in the storage backend and returns T and n to the client facing server. The secret token T is not stored by the CTA. The client facing server can now use the the secret token n number of times. This process is detailed in algorithm 1.

Hash chain verification: Hash chains are used to authenticate client facing server requests which require access to a sensitive resource. The hash chain verification is detailed in algorithm 2.

Data: Hash Chain secret T and Hash chain length N

Result: Hash Chain $H^N(T)$

$i \leftarrow 1$;

$H^1(T) \leftarrow H(T)$;

while $i <= N$ **do**

  $i \leftarrow i + 1$;

  $H^i(T) \leftarrow H(H^{i-1}(T))$;

end

return $H^i(T)$ *where* i *equals* N ;

Algorithm 1: Generating a Hash Chain

Data: Authentication key $H^{i-1}(T)$ from the client

Result: Response from sensitive resource

Let $H^i_{client} = H(H^{i-1}(T))$;

Let $AuthenticationData = fetch(H^i_{client})$;

if *AuthenticationData exists in storage backend* **then**

  replace H^i_{client} with $H^{i-1}(T)$ in storage backend ;

  fetch and return response from sensitive resource ;

else

  return authentication failure ;

end

Algorithm 2: Verification of Hash Chain authentication

Table 4.1: List of Python libraries used

Library == Version	Purpose
tornado==4.4.2	Asynchronous networking library used to implement the CTA and proxy requests to sensitive resources
peewee==2.8.1	An ORM used to implement the DB back-end for
redis==2.10.5	A Python library to communicate with Redis key value store
funkload==1.17.2	A load testing library we used when testing performance overhead
simpy==3.0.10	A Python discrete event simulation library
pandas==0.19.2	A Python data analysis library
numpy==1.11.3	A scientific computing library for Python
matplotlib==2.0.0	A 2D plotting library [25]

4.2 Implementation

We implemented prototypes of our architecture using the Python programming language version 3.6. We chose Python over other programming languages such as Java, C / C++, and Go for 2 reasons. The first one being our familiarity with the language and its ease of use which enabled us to quickly iterate through prototypes. The second reason was the availability of libraries. Python has a thriving community of developers who have written libraries to assist in implementing many of our needs. A full list of libraries can be found in table 4.1.

The most significant disadvantage of our choice of using Python was performance. Python as a dynamically typed language has a significant overhead when compared to compared to compiled and statically typed languages such as Java, and C++. This disadvantage was primarily noticeable in our simulations as we were unable to use native threads to parallelize our simulations due to Python's Global Interpreter Lock

(GIL). We were able to overcome this using the *multiprocessing* module from the Python standard library which allowed us to parallelize our simulations using multiple processes instead of thread. Multi-process parallelization has an additional overhead of serialization and de-serialization when collecting the results of our simulation. We did not notice experience any significant impediments due to the serialization overhead.

4.3 Simulations

Simulation is the process of attempting to duplicate and imitate a particular real-life scenario. We were required to simulate the interactions between a cloud system and its adversary. Our goal with these simulations was to analyze the effects of an artificially limited server lifetime and how such a limited lifetime could adversely affect an attacker.

4.4 Discrete event simulation

Discrete event simulation, is a type of simulation where a system is modeled as a series of events which occur at distinct points in time. A system can contain multiple different parts. Different parts of the system can then be modeled to react to certain events. Such reactions could potentially trigger other events as the system progresses through time. The primary assumption in a discrete event simulation is that no change occurs between events. Thus the simulation program can quickly move through all events which get triggered over the course running a simulation. Upon analyzing various mechanisms of simulating computer systems we concluded that discrete event simulations would be ideal for our purposes.

Discrete event simulations are in contrast to continuous simulations which attempt to simulate continuous functions using real numbers. A classic examples of a continuous simulations are weather simulations and rocket trajectory simulations. Unlike a discrete event simulation where a system's behavior changes only in response to a fi-

nite number events over time, a system's behavior changes perpetually in a continuous simulation based on one or more continuous functions.

Since most computer systems and networks are designed to respond to specific events, discrete event simulation was the ideal choice to model the interactions between an attacker and a cloud system. Our choice is further supported by a study conducted to compare Discrete and Continuous simulations [61]. The authors of the study conclude that discrete event simulations are mainly suited towards finding statistical estimates for equilibrium values. Since finding statistical estimates is the primary goal of our simulations we decided to use discrete event simulations to model our proposed architecture.

Upon deciding to build discrete event simulations, we explored various frameworks and libraries such as SIM.JS[54], Facsimile [13], and SimPy [41]. We decided to use SimPy as the Python programming language offered tools such as NumPy, Pandas, and Matplotlib for data analysis and generating charts. This choice allowed us to streamline our work-flow to execute a simulation, processes the obtained result and generate a plot using a single integrated program.

4.5 Game definition

Our game consists of two players. The defender, who in real life scenarios would be the cyber-security team for an organization, and the malicious actor who is working towards compromising the cloud infrastructure of the organization.

The malicious actor has two primary strategies, to attack by trying to compromise the system, or refrain from attacking. The malicious actor gains a pay-off only when they successfully compromise a system. Along with the pay-off benefits attacking a system has costs associated with it. Costs such as bandwidth, C&C [1]server costs and Cost of being discovered.

[1]C&C server refers to Command and Control Server

The defender's strategies are to perform a scan and increase / decrease the hash chain length or not perform a scan. Scanning is considered to be an expensive operation. This implies that a scan cannot be performed on all CFS instances. A technique similar to packet sampling [16] can be used to sample CFS instances for analysis. Such a sampling technique merely implies that attacks are detected with a probability of $P(D)$. This scanning strategy is highlighted in figure 4.2.

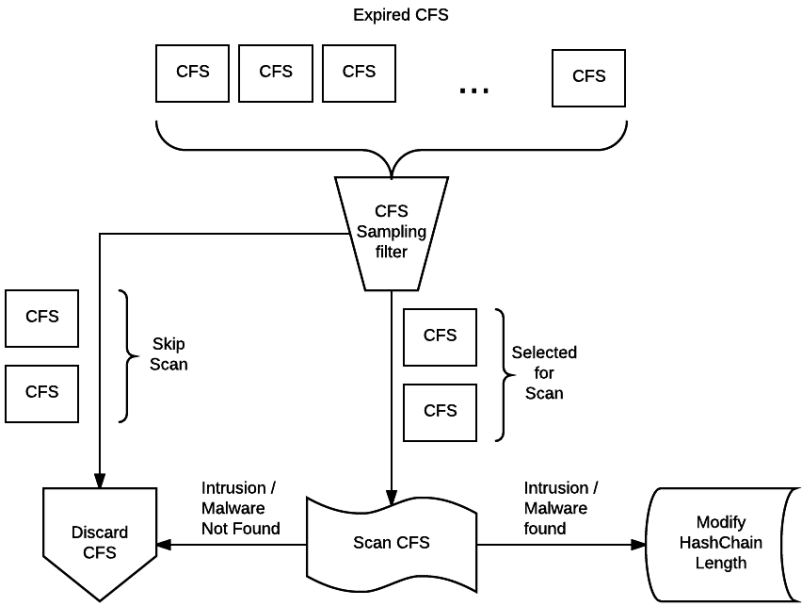

Figure 4.2: CFS scan strategy based on sampling

Though TTC value could vary significantly for each system we've chosen data from a large scale study [24] conducted at Sweden as a baseline to evaluate and model our system.

After some initial testing we realized that the scanning strategy, though realistic in nature, merely increased the complexity of our game. So we switched to a simpler 2 player game with an attacker and a defender. The attacker's strategies remain un-

changed, with the attacker having the ability to attack or not attack. The defender's changes to either reset or not reset, with reset referring to terminating a CFS. In section 5.5 we solve for the mixed strategy Nash Equilibrium of this game.

Chapter 5

RESULTS OF A CASE STUDY

In this chapter we present the results of a case study. We describe three types of results, Performance testing, Simulations and Game theoretic analysis. Our aim with these results is to showcase a framework for evaluation of MTD architectures which utilize limited server lifetimes.

5.1 Performance

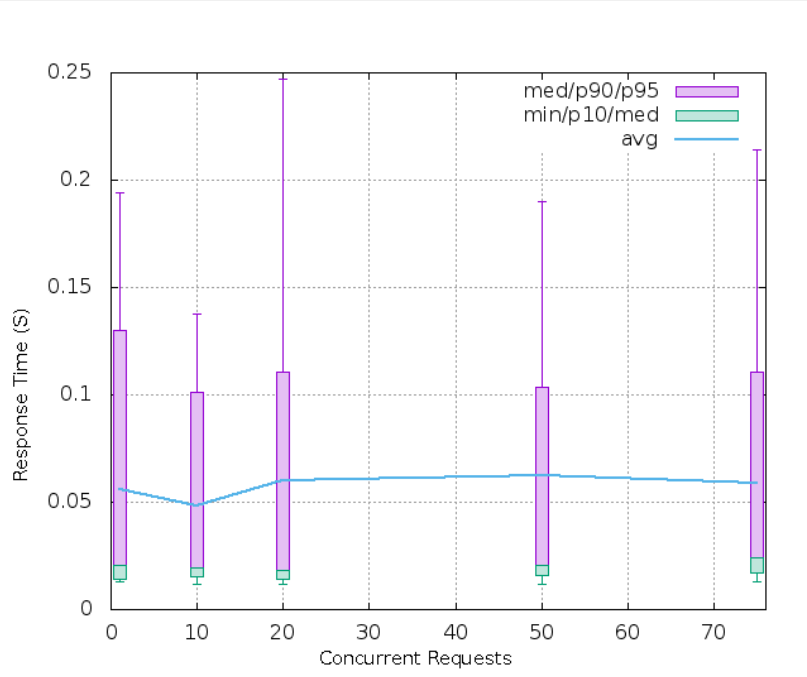

Figure 5.1: Baseline with non authenticating proxy

One of the key areas of concern when implementing a Moving Target Defense

(MTD) system is performance. Based on our architecture we decided to evaluate performance as a function of overhead. Overhead in this scenario can be defined as the additional resources needed to implement our MTD architecture. Performance of an architecture is inversely proportional to overhead. This implies that a higher overhead could potentially lead to lower performance, if the hardware used is unable to support such an overhead.

To quantify overhead, we compare the response times of a server setup which uses an authenticating CTA Proxy, with a server setup which does not uses an open CTA Proxy. The authenticating CTA Proxy issues, and validates hash chains, while the open CTA proxy does not require any authentication information.

Figures 5.1 and 5.2 illustrate the average response times when communicating through a proxy. The X-axis is the number of concurrent requests made and the

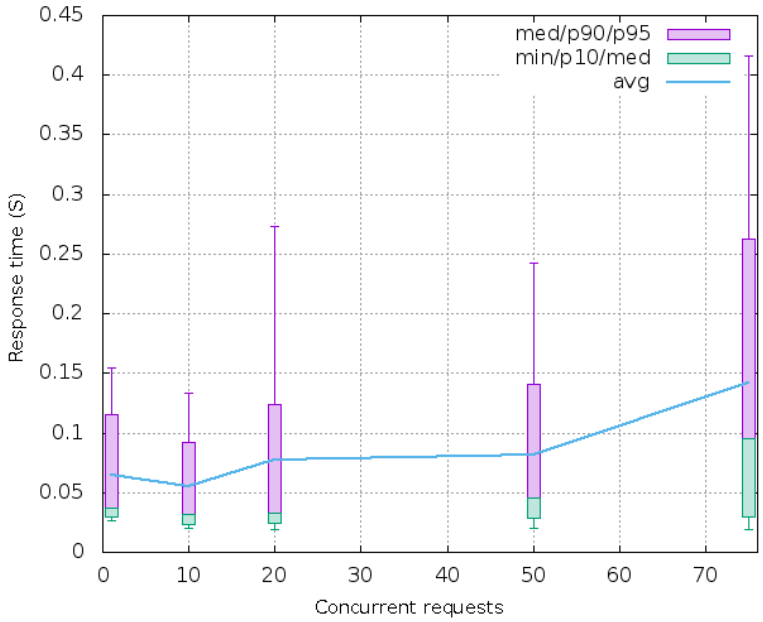

Figure 5.2: Overhead observed with an authenticating proxy

Y-axis describes the mean response time in seconds. Figure 5.1 is the baseline measurement and shows us that the mean response time with a simple non authenticating proxy is around 0.05 seconds. Figure 5.2 showcases the mean response times of an authenticating proxy (see chapter 4). Note that the Y-axes are scaled differently in figures 5.1 and 5.2.

Our initial goal was to illustrate an average overhead of no more than 0.05 seconds. As figures 5.1 and 5.2 illustrate, the response overhead, though slightly higher than our expectations, is still reasonable for a proof of concept implementation.

5.2 Effects of TTL on successful attacks

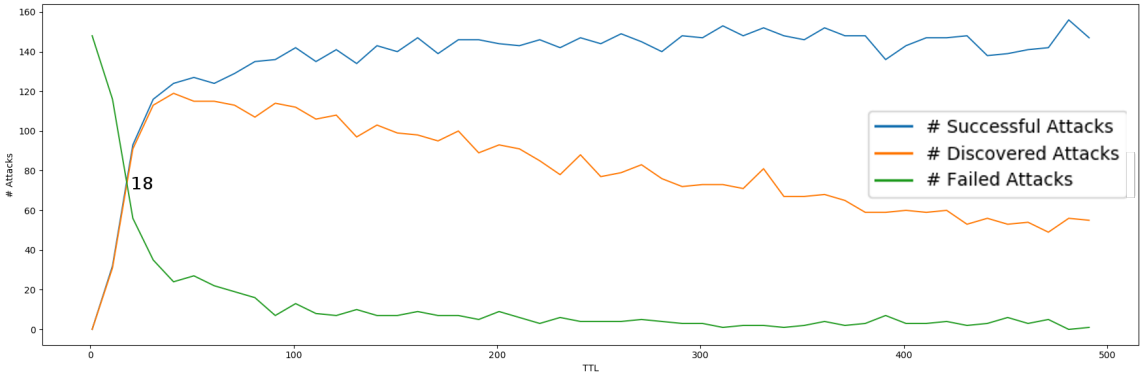

Figure 5.3: Baseline simulation for an adaptive algorithm

Listing 5.1: Naive adaptive algorithm

```
# ATTACK_THRESHOLD is mean tolerable attack rate
# MIN_TTL a minimum value for TTL to prevent current_ttl <= 0
if (attack_rate_based_on_scans > ATTACK_THRESHOLD and
    current_ttl > MIN_TTL):
    current_ttl -= 1
    scan_probability += 0.5
```

```
else:
    current_ttl += 1
```

The aim of this discrete event simulation was to illustrate the effects of an adaptive algorithm against a naive attacker who attacks at a constant rate. The adaptive algorithm (listing 5.1) monitors the attack rate and reduces the effective TTL of a CFS when the attack rate increases. An attack is successful in this simulation when the attacker has enough time to perform reconnaissance, exploit a vulnerability, and have a pre-defined minimum amount of time left over in the CFS's TTL to effectively utilize the server's resources. If any of these criteria are not met the attack is considered a failure.

Figures 5.3 and 5.4 compare the effects of using an adaptive algorithm vs not an adaptive algorithm. The X-axis is the measurements taken at various TTLs and the Y-axis depicts a count of attacks. Each colored line on figs. 5.3 and 5.4 represent different types of measurement. The blue line represents a count of successful attacks, while the orange line depicts the number of successful attacks which were discovered using the scanning strategy described in section section 4.5, and the green line presents the number of failed attacks. The simulation ran for $10,000$ units of Simulation time

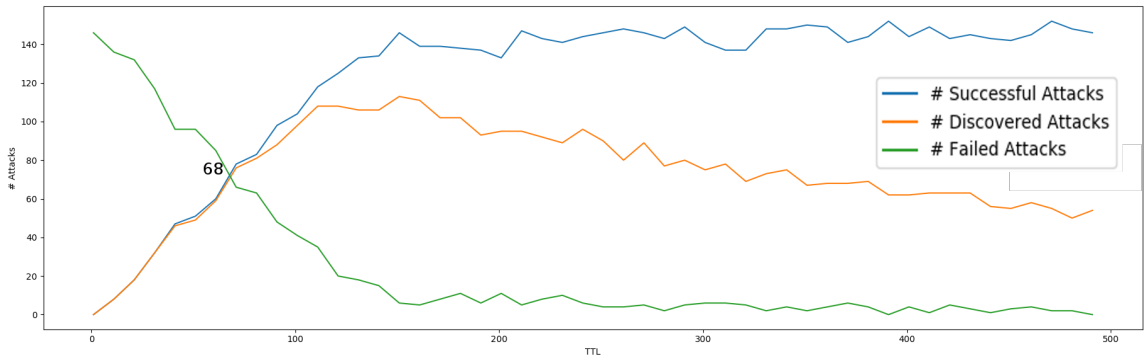

Figure 5.4: Adaptive algorithm simulation

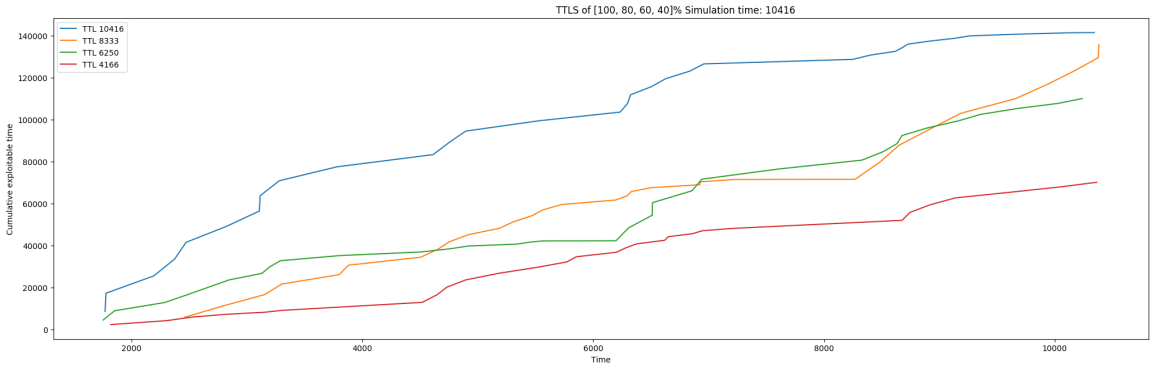

Figure 5.5: Cumulative exploitable time for TTLs of 100%, 80%, 60%, and 40% simulation time

for each point on the X-axis. TTL values ranging from 1 to 10,000 were simulated in increments of 100.

By comparing figures 5.3 and 5.4 we can conclude that our naive adaptive algorithm increases the effectiveness of a limited lifetime server in reducing the number of successful attacks, especially with lower values of TTL. This simulation inspired us to simulate a Nash Equilibrium attacker (see section 5.6) and modify our naive adaptive algorithm to be based on the defender's Utility rather than the rate of attack.

5.3 Exploitable time

5.3.1 Background

Exploitable time is the amount of time that an attacker has access to a cloud system after the system has been successfully compromised. Measuring exploitable time enables a comparison of the value that an attacker can obtain as a result of compromising our system. Value for an attacker could be quantified using many different parameters. For example, an attacker could potentially hijack computational resource from our machines by installing malware and selling server time to bot nets. Another example would be stealing sensitive information such as database records by connect-

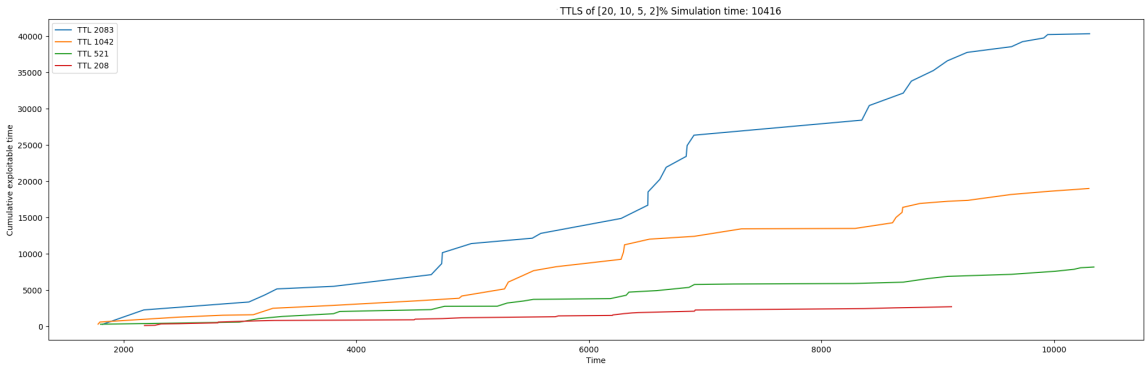

Figure 5.6: Cumulative exploitable time for TTLs of 20%, 10%, 5%, and 2% simulation time

ing to a database server. In all these examples we note that the value an attacker can derive from a system is in direct co-relation to the amount of exploitable time that is available. Thus we can infer that cumulative exploitable time can be used as a precursor to computing other much more useful metrics which aid in evaluating the overall security and exploit-ability of a cloud system.

The simulation setup for this simulation is equivalent to the setup described in section 4.4.

5.3.2 Analysis

Figures 5.5 to 5.7 describe the cumulative exploitable time that an attacker can potentially use. The X-axis shows us the progression of simulation time and the Y-axis shows us the cumulative sum of exploitable time. Exploitable time is computed based on simulation time and is not based on any real units such as seconds.

The Figures 5.5 to 5.7 illustrate that as we decrease the maximum lifetime of a server (TTL) the cumulative exploitable time decreases significantly. For example there is an order of magnitude difference between the final cumulative exploitable time of a system with TTL 2083 and a system with TTL 208. This leads us to conclude

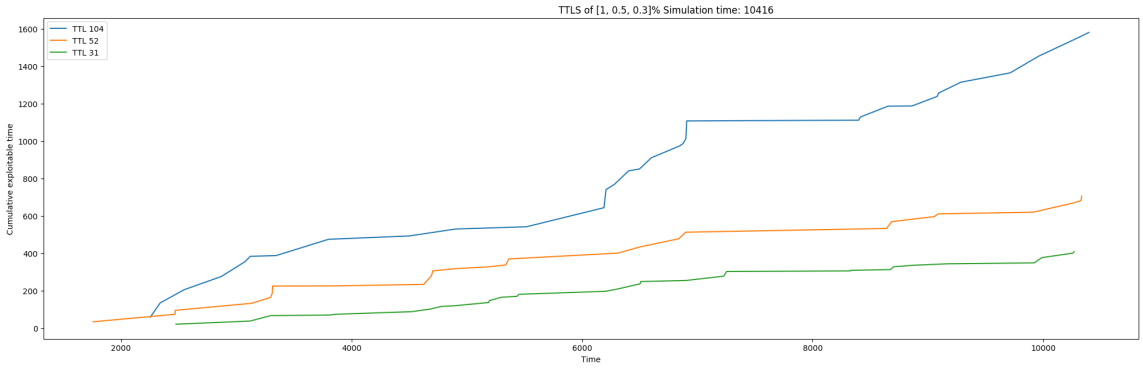

Figure 5.7: Cumulative exploitable time for TTLs of 1%, 0.5%, and 0.3% simulation time

that a decrease in TTL significantly decreases the value that an attacker can derive from our system.

Figure 5.8 visualizes the mean exploitable time for each value of TTL. The Y axis represents the exploitable time and each bar describes the mean exploitable time for its corresponding value of TTL. Lower exploitable time is better as the attacker has less time to access the system. As we can see the mean exploitable time decreases significantly as TTL decreases.

5.4 Quantifying cost being exploited

5.4.1 Background

To demonstrate a financial value in using limited lifetimes to secure Client Facing Servers (CFS) we ran a simulation experiment to quantify the effect of TTLs on the cost to the defender of being exploited by an attacker. To obtain concrete values for quantifying the effect of TTLs we relied on a regression equation by Romanosky et al.[47], in which the authors assign a dollar value to the cost of being attacked as a

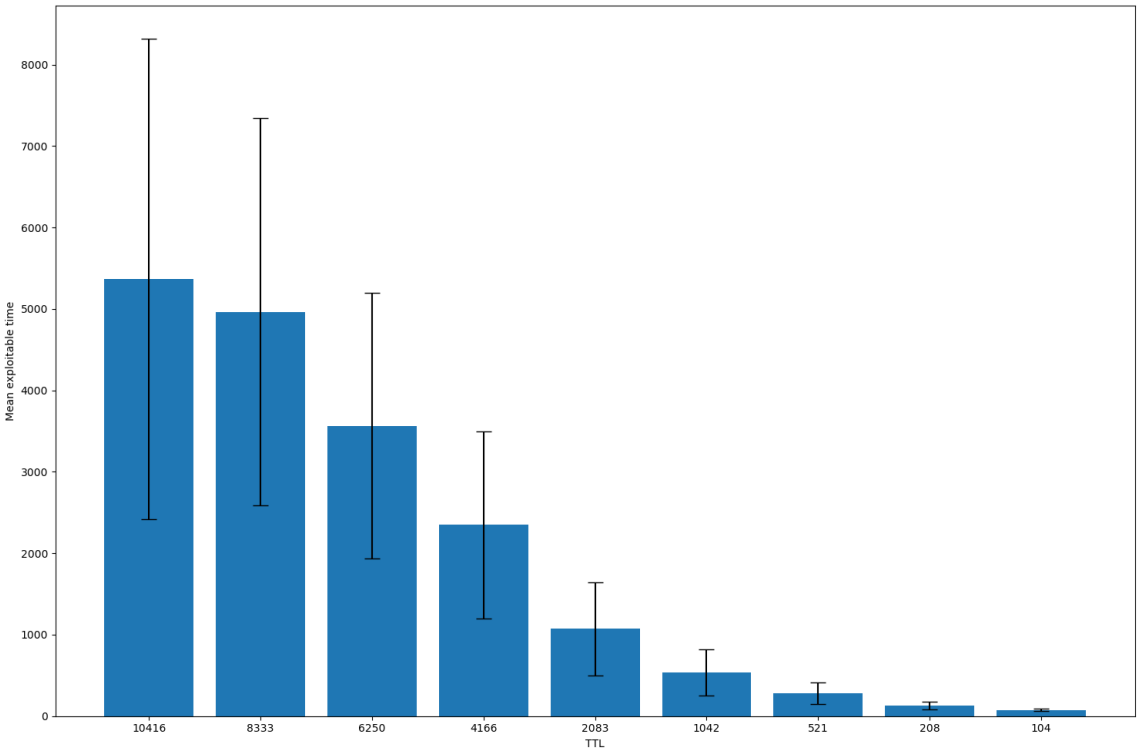

Figure 5.8: Mean exploitable time and its standard deviation (Lower is better)

function of the amount of information which was stolen by the attacker.

$$Log(impact) = 7.68 + 0.76 * Log(records) \tag{5.1}$$

In the equation presented by Romanosky (see eq. (5.1)), *records* refers to the number of sensitive records that an attacker is able to steal from the defender, while *impact* refers to the total loss in dollars that an organization could potentially experience as a result of being exploited by an attacker. We use the simulation data presented in section 5.3 to obtain a value for the number of *records* by multiplying a constant factor with the cumulative exploitable time. This constant factor is considered to be the number records that an attacker can steal in unit time. For the purposes of this simulation we consider the constant factor to be 5.

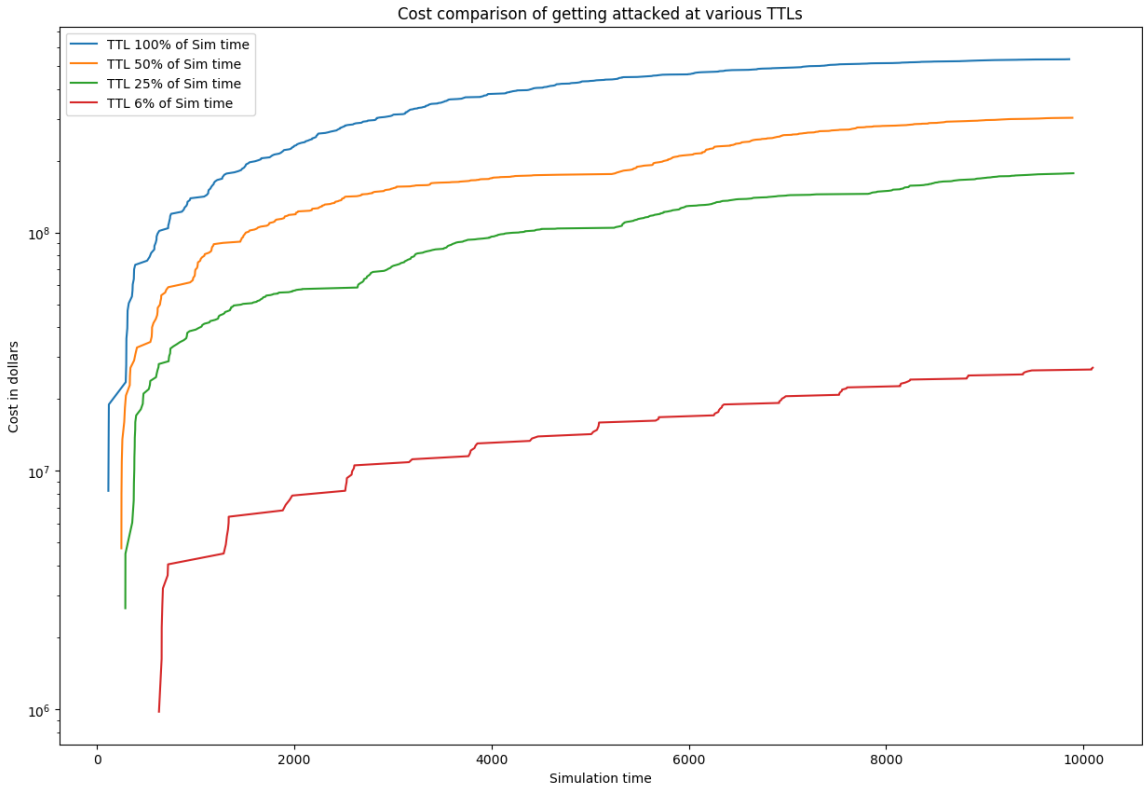

Figure 5.9: Cost of being exploited

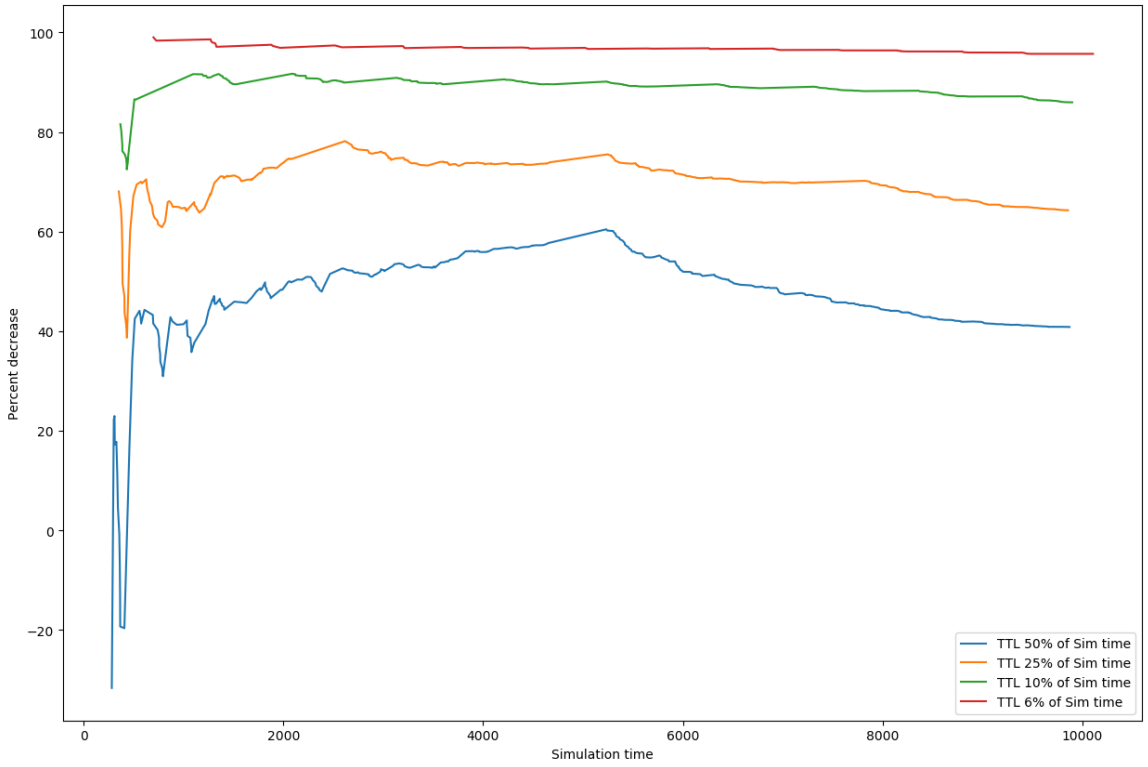

Figure 5.10: Percent difference of cost of being exploited over baseline TTL

5.4.2 Analysis

Figure 5.9 compares the cost of being attacked by a naive attacker at various life times (TTLs) of a Client Facing Server (CFS). The X-axis depicts the passage of time within the simulation and the Y-axis represents the cumulative cost in dollars. Each colored line on the graph represents a particular TTL as a percent of of the total simulation time.

In fig. 5.9 each line represents a particular value of server lifetime (TTL). The baseline value is a TTL of 100% simulation time which implies that a CFS's lifetime tends to infinity within the simulation. The X-axis in the figure depicts the passage of simulation time and the Y-axis is a log-scaled representation of the cumulative cost of being attacked by applying eq. (5.1) as described in section 5.4.1. From this figure

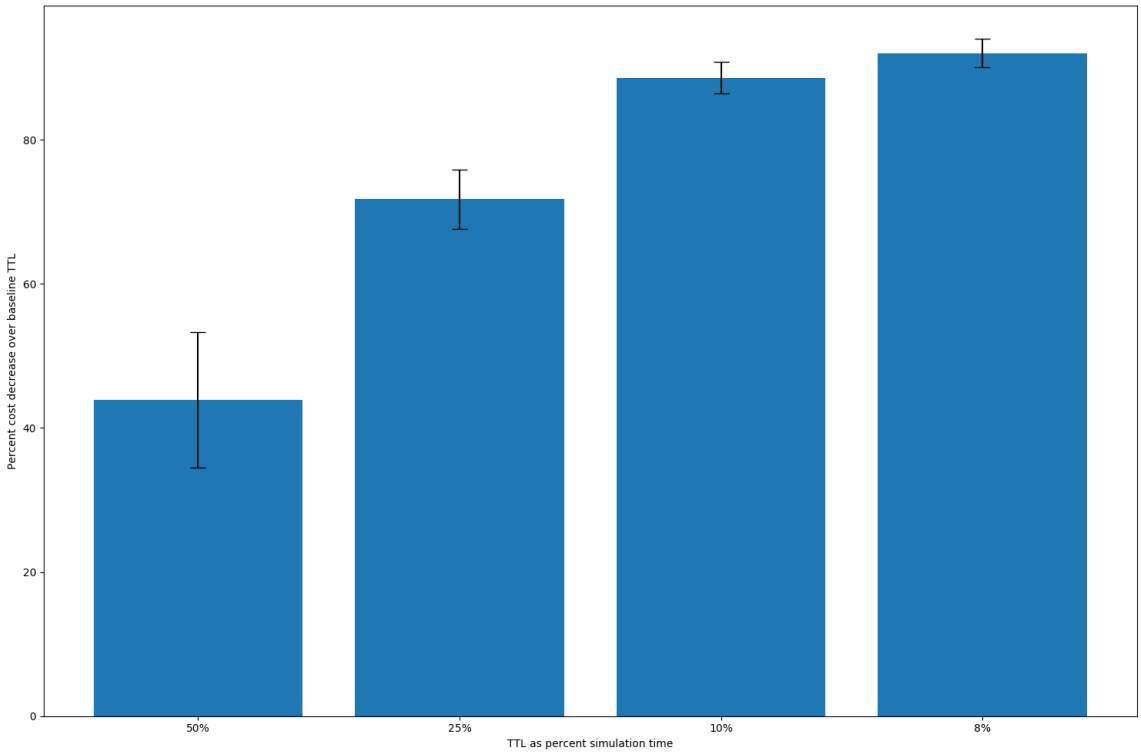

Figure 5.11: Percent difference in cost of being exploited over baseline TTL (Higher is better)

we can infer that the there is a significant reduction in the cost of a data breach when using a smaller TTL value as compared to a higher TTL value. For example, compared to the baseline (100% simulation time), a TTL of 6% (of simulation time) decreases the cost of a data breach by approximately 90% as can be seen in fig. 5.10. Figure 5.11 showcases this data as a bar chart with a baseline of 100% simulation time.

5.5 Nash Equilibrium

In this section we solve for the mixed strategy Nash Equilibrium of the game which was defined in section 4.5

Defender \ Attacker	Attack	No Attack
Reset	$B_{att} * ET - C_{att} - C_{recon}$ $B_{use} * TTL - C_{vic} - C_{reset}$	$-C_{recon}$ $B_{use} * TTL - C_{reset}$
No Reset	$B_{att} * MET - C_{att} - C_{recon}$ $B_{use} * MAXT - C_{vic}$	$-C_{recon}$ $B_{use} * MAXT$

Figure 5.12: Payoff Matrix with formulations for each payoff

Defender \ Attacker	Attack	No Attack
Reset	-4.243 -105.047	-1 24.953
No Reset	0.94 -46872	-1 5208

Figure 5.13: Payoff Matrix for computing mixed strategy Nash equilibrium

5.5.1 Background

Examining our architecture using game theory's lens allowed us to model the payoffs and motivations of the attacker and defender. Using such a model allowed us to compute values of Utility at any given moment in time. The advantage of using utility instead of attack rate when adapting our system is discussed in section 5.6.1.

Table 5.1: Payoff for each scenario in payoff matrix (see fig. 5.12)

Player	Attacking ?	Resetting ?	Payoff
Attacker	Yes	Yes	$B_{att} * ET - C_{att} - C_{recon}$
Defender	Yes	Yes	$B_{use} * TTL - C_{vic} * ET - C_{reset}$
Attacker	No	Yes	$B_{att} * 0 - C_{att} * 0 - C_{recon}$
Defender	No	Yes	$B_{use} * TTL - C_{vic} * 0 - C_{reset}$
Attacker	Yes	No	$B_{att} * MET - C_{att} - C_{recon}$
Defender	Yes	No	$B_{use} * MAXT - C_{vic} * MAXT$
Attacker	No	No	$B_{att} * 0 - C_{att} * 0 - C_{recon}$
Defender	No	No	$B_{use} * MAXT - C_{vic} * 0$

Nash Equilibrium attacker

A Nash Equilibrium attacker is a malicious entity who is attempting to exploit the defender's system at a constant rate. This constant rate is defined by the mixed strategy Nash Equilibrium of the system as a whole. We modeled our system as a game in an attempt to quantify this concept of a rational attacker (refer Rational player). To quantify a rational attacker we defined a pay-off matrix and solved for the mixed strategy Nash Equilibrium. The solution to the mixed strategy Nash Equilibrium allowed us to obtain the attack probability of a rational attacker.

5.5.2 Solving for Nash Equilibrium

To solve for the mixed strategy Nash Equilibrium we defined a Payoff Matrix as shown in fig. 5.12 and elaborated in table 5.1. We then proceeded by setting the expected payoff for each action that a player could take, equal to the payoff for the alternative action. Solving these equations we obtained the probability of attack, defined by

Table 5.2: Frequently used notation and its during the experiment

Notation	Value	
$B\_\{att\}$	0.0005	Benefit of a successful attack
$B\_\{use\}$	0.5	Benefit of server utilization
$C\_\{att\}$	3.256	Cost of mounting an attack
$C\_\{recon\}$	1	Cost of reconnaissance
$C\_\{reset\}$	0.047	Cost of resetting a server
$C\_\{vic\}$	5	Cost of being a victim
ET	$TTL - AT$	Exploitable time
MET	$MAXT - AT$	Max Exploitable Time
$MAXT$	$434 * 24$	Max Time
TTL	50	Server lifetime
AT	24	Attack time
P_{att}	see eq. (5.2)	Probability of attack
P_{reset}	see eq. (5.3)	Probability of reset

eq. (5.2), and probability of reset defined by eq. (5.3).

$$P_{att} = \frac{C_{reset} + (B_{use} * MAXT) - (B_{use} * TTL)}{(C_{vic} * MAXT) - (C_{vic} * ET)} \tag{5.2}$$

$$P_{reset} = \frac{C_att - B_att * MET}{B_att * ET - B_att * MET} \tag{5.3}$$

Upon solving eq. (5.2) using the values specified in table 5.2 we get a value of 0.099 or about 9.9% for P_{att}. This means that a rational attacker would attempt to exploit a CFS with a probability of 9.9% at any given point in time.

5.6 Adapting to a Nash Equilibrium attacker

Listing 5.2: Utility based adaptive algorithm

```
# We used MIN_ACCEPTABLE_UTILITY = 2
# and MAX_ACCEPTABLE_UTILITY = 20
# utility_change is the mean difference in utility
# over the last 100 time units.
if current_utility < MIN_ACCEPTABLE_UTILITY:
    new_ttl = current_ttl - abs(round(utility_change * 0.1))
elif current_utility > MAX_ACCEPTABLE_UTILITY:
    new_ttl = current_ttl + abs(round(utility_change * 0.1))
else:
    new_ttl = current_ttl
update_ttl(new_ttl)
```

5.6.1 Background

We modified the naive adaptive algorithm from section 5.2 to better suit a Nash Equilibrium attacker. A Nash Equilibrium attacker is an attacker who's attempting

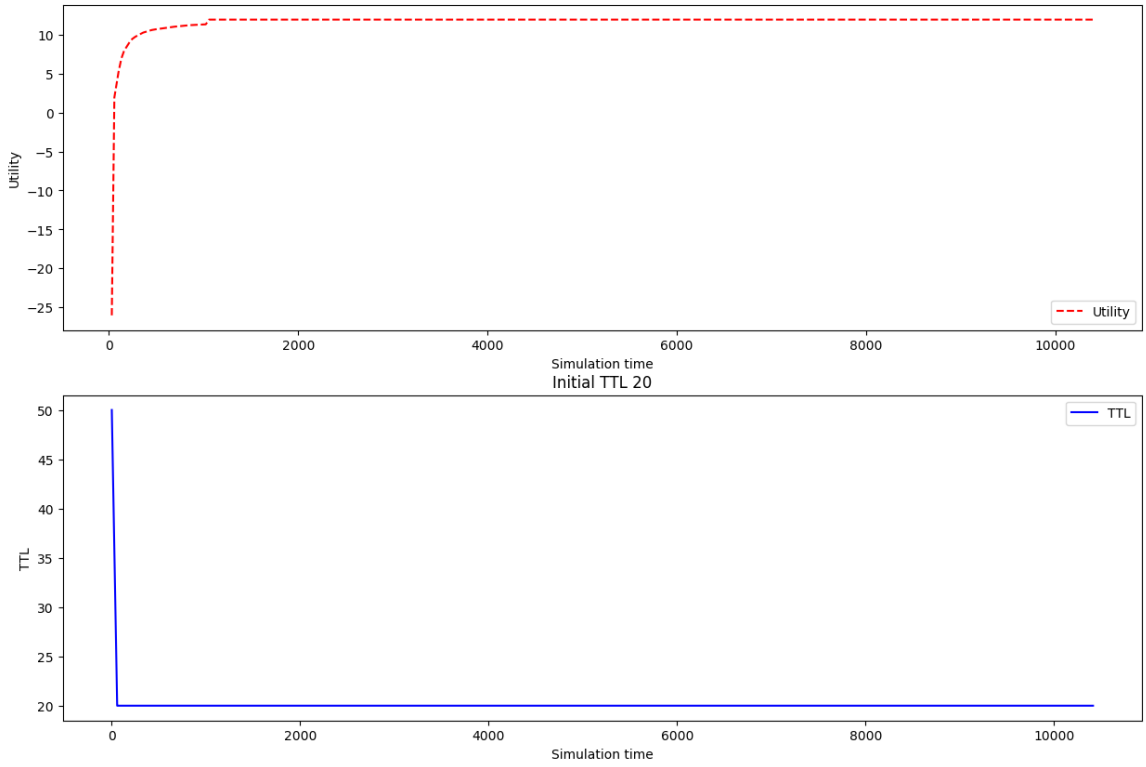

Figure 5.14: Comparing the progression of utility based adaptive algorithm with an initial TTL of 20

to exploit the defender's system at a constant rate as described in section 5.5. We updated our algorithm (listing 5.2) to factor in utility change rather than attack rate. This change offers the algorithm the ability to adapt to multiple factors beyond merely the attack rate. Utility is computed as shown in eq. (5.4).

$$U_{def} = (TTL * B_{use}) - C_{res} - (T_{exp} * C_{vic}) \tag{5.4}$$

5.6.2 Analysis

Figures 5.14 to 5.18 illustrate the effects of applying the adaptive algorithm described by listing 5.2. In each figure, the X-Axis represents simulation time and the Y-Axis

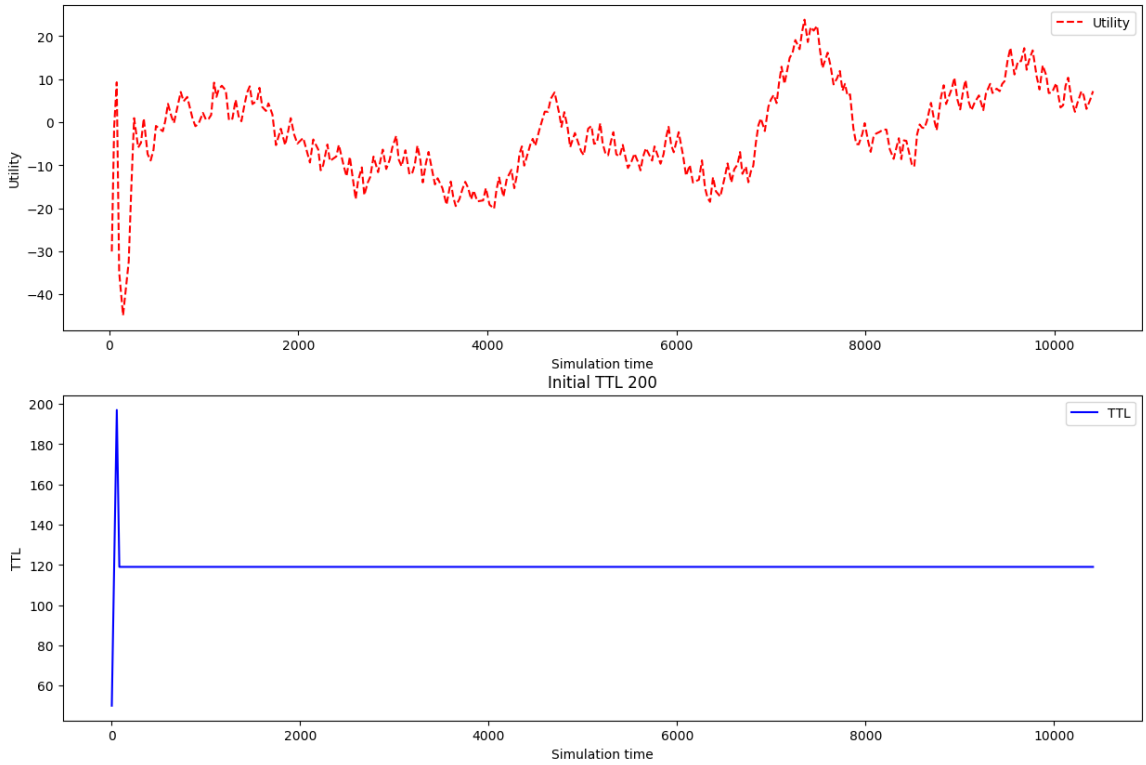

Figure 5.15: Comparing the progression of utility based adaptive algorithm with an initial TTL of 200

represents Utility, and TTL.

In each of these simulations illustrated by figs. 5.14 to 5.18 we see that the algorithm (listing 5.2) quickly moves into the acceptable utility range by increasing or decreasing TTL. We observe that with higher initial TTLs, such as 5000 units of simulation time. In our testing we noticed that the adaptive algorithm described by listing 5.2 falters as we approach an Initial TTL equivalent to 50% of simulation time. This effect can be seen in fig. 5.18 where Utility and TTL fluctuate vastly as time progresses. We believe the cause of this to be the delay in updating TTLs as servers have a longer lifetime.

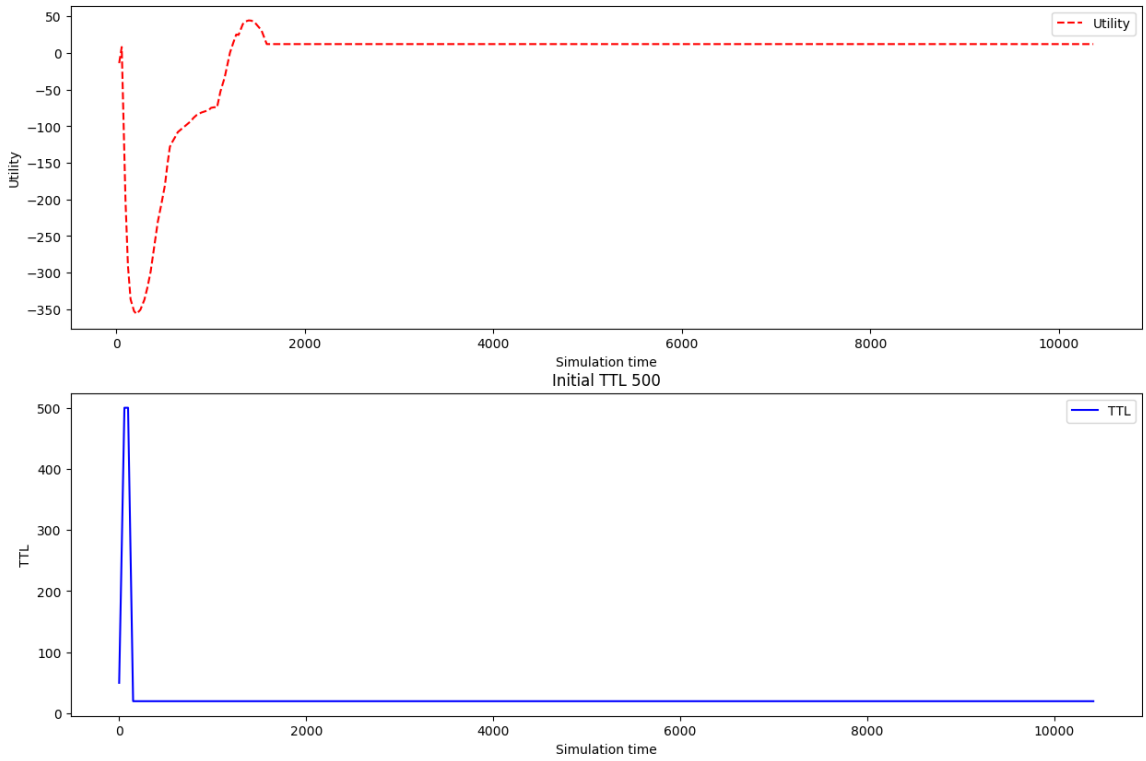

Figure 5.16: Comparing the progression of utility based adaptive algorithm with an initial TTL of 500

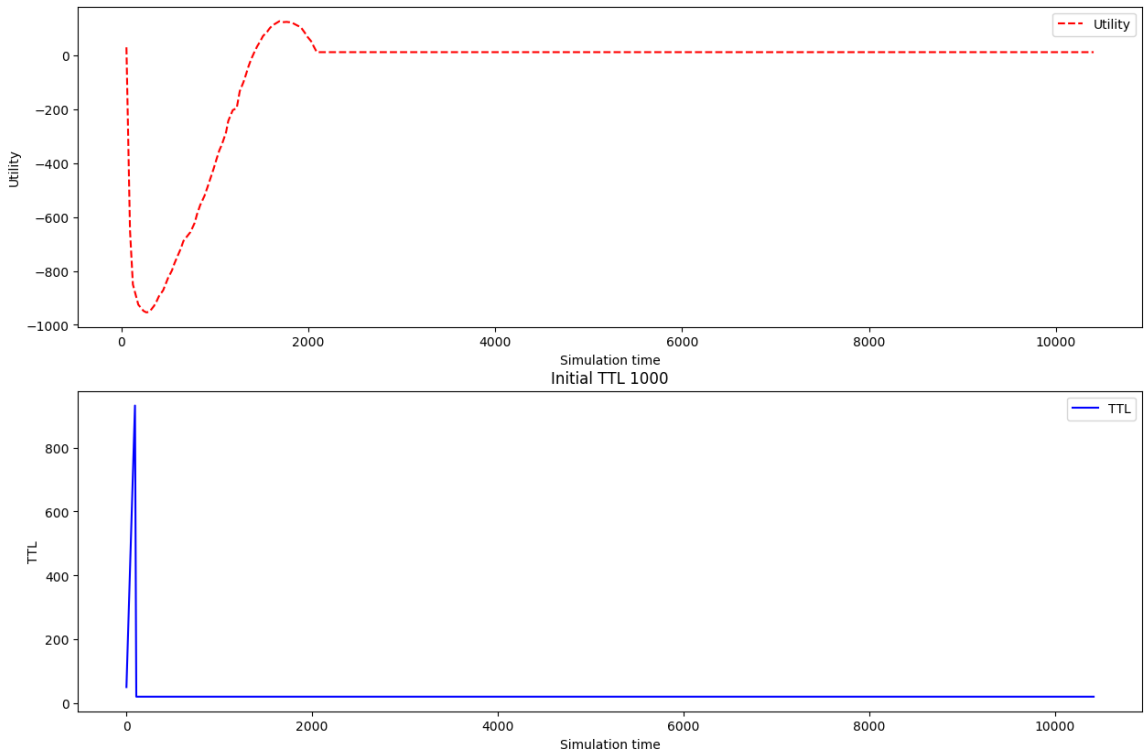

Figure 5.17: Comparing the progression of utility based adaptive algorithm with an initial TTL of 1000

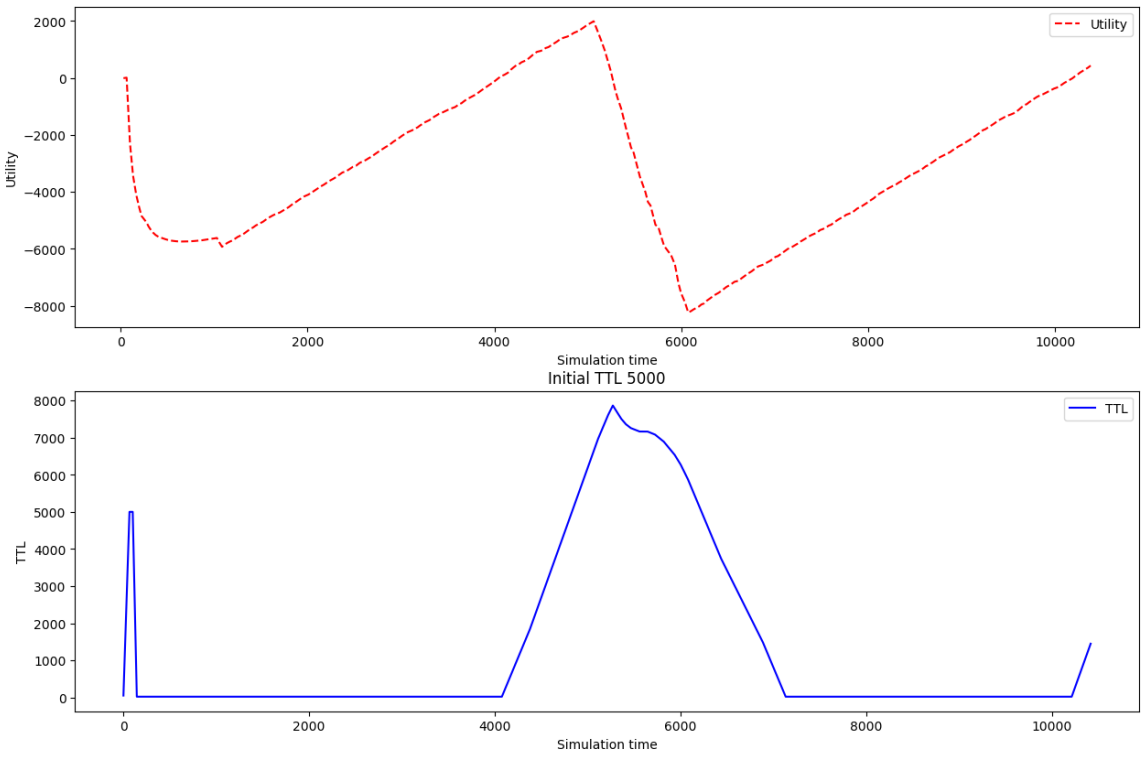

Figure 5.18: Comparing the progression of utility based adaptive algorithm with an initial TTL of 5000

Chapter 6

DISCUSSIONS

6.1 Security Threats

Our proposed architecture of using a CTA to proxy requests on behalf of all the clients places the CTA as a single point of failure. We deem this an acceptable trade-off as the CTA is not a public facing system and only serves the purpose of verifying hash chains and proxying requests. As a result, hardening its defenses against possible threats is relatively easier when compared to hardening a larger number of CFS to an equal measure.

6.1.1 Malicious Request Proxy

The request proxy component acts on behalf of the client facing server to make a call to a sensitive resource such as an API endpoint. The request proxy is also responsible for attaching authentication information such as an API Key when contacting the API endpoint. Upon receiving a response from the sensitive resource the request proxy strips out sensitive information such as API Keys and Refresh Tokens before forwarding the response to the client facing server. Under this threat scenario the request proxy is assumed to be untrustworthy and potentially malicious despite the our earlier argument that this case is unlikely.

Chen et al [17] propose a solution to this problem of a Malicious proxy using trusted hardware such as Trusted Platform Module (TPM) or the IBM 4758 cryptographic co-processor [44]. The CTA executable would also verified by a trusted third party to operate correctly as a proxy as described in [44].

6.1.2 MTD in our proposed architecture

As identified by Green et al [22] and highlighted in section 3.3 our architecture provides Moving target defense in the following ways.

Unpredictability: An attacker is unable to predict the TTL of a CFS even after repeated observations as the TTL is varied using one of two methods. The first method of using hash chains ties the lifetime of a CFS to an unpredictable request rate. While the second method of adaptive algorithm (see chapter 5) ties a server's TTL to an unpredictable rate of attacks. When a defender uses one or both of these methods, an attacker would ideally experience a disadvantage when increasing their rate of attacks.

Another source of unpredictability are Cryptographic hash functions [46]. They are designed to be collision resistant and comply with the avalanche effect and thus the generated sequence is highly unpredictable in nature. Hash chains can compound this effect by sequential application of the cryptographic hash function.

Vastness: A secure cryptographic hash function such as SHA-256 has a vast state space as the output could be any of 2^{256} different values. As a result, the likelihood of an attacker guessing the correct value is negligible.

Revocability: The proposed architecture applies a finite lifetime for each CFS. Upon TTL expiry a server could be considered revoked. We postulate that automated and unpredictable revocation decreases the attack desirability of an implementation of our architecture. Another source of revocability are Hash Chains,individual chains or a group of chains to be revoked merely by purging them from the database. Further, group, or hierarchical revocability can be achieved using Merkle hash tree implementations as a means of Hash chain generation.

6.2 Limitations

In this section we discuss the various limitations and shortcomings of our research.

6.2.1 Simulations are imitations

Our simulations, like most simulations, are simplified versions of real life scenarios. The goal of these simulations was to serve as a proof of concept under conditions isolated from the unpredictability of real world. For example we vastly simplify an attackers ability to exploit a system using a probability of attack (P_{att}) and a fixed span of attack time. Such a simplification may not apply to real life scenarios, especially when an attacker exploits zero-day vulnerabilities. If an attacker exploits zero-day vulnerabilities the attacker may be able to gain near instantaneous access to a CFS though the attacker would still be limited by the TTL.

Unpredictable instance start up latencies common in cloud infrastructure and our simulations do not account for these latencies. A common solution for handling start up latency is to over-provision server infrastructure or utilize a server pool[32]. Research into reliability engineering and the usage of spot instances [58, 45] also offers multiple solutions to the problem of reliability with cloud VMs with limited lifetime.

In our simulation on the effects of TTL on the number of successful attacks (see section 5.2) we utilized a probability of attack (0.9%) computed using the the values of TTFC and TBC from the large scale study[24] which was discussed in section 3.5. This value may not represent a real life probability of attack as the study was conducted on windows desktop computers in a large organization, this is unlike most cloud infrastructure deployments which are Linux based [8].

6.2.2 Not a complete solution

Cloud computing requires a holistic approach to security with multiple layers [15, 43] and our proposed architecture is intended to be one of those layers. We strongly discourage anyone from utilizing our architecture as the only layer in a security setup.

6.2.3 Large cloud deployments

We intend our architecture to part of a multi-tired security system within a large organization with a dedicated security team. A small business running a few servers on AWS may not see any gains by implementing our architecture. Smaller organizations are better served by fixing low hanging issues such as bugs and vulnerabilities listed in OWASP Top 10 [57, 3].

6.3 Future work

The primary avenue of future work is implementing a honeypot to test the validity of our simulations. Such an implementation would ideally contain a prototype version of our architecture which has been instrumented to monitor the operating system for any changes in disk, memory, or processes. The honeypot could also be used to study the effects of various adaptive mechanisms which could be implemented as part of our architecture. Adaptive mechanisms could also be developed using machine learning models to account for multiple factors and dynamically learn based on the environment. Such an adaptive system would be able to dynamically set the TTLs to optimize for reducing attacker utility without compromising value for the defender.

Another avenue of exploration would be evaluating and testing CTAs implemented as distributed entities such as DHTs and blockchains. We postulate that such a system would be much more resilient to compromise even if parts of a CTA are exploited by an attacker.

Chapter 7
CONCLUSIONS

In this thesis, we proposed a cloud architecture which provides a layer of Moving Target Defense. Our architecture achieves moving target defense by limiting server lifetimes to a fixed time period. By limiting the lifetime of servers we also reduce the amount of time that an attacker has access to a server. We tie the lifetime of server to factors which are difficult to predict by an attacker such as request rate and utility. By tied the lifetime of servers to difficult to predict factors, we increased the uncertainty and apparent complexity of our system to achieve Moving Target Defense.

We used simulations to perform a case study evaluation of our architecture. Our simulations show that our architecture vastly reduces the cost of being exploited by reducing an attacker's access to total exploitable time. Finally we modeled our system as a game to solve for its mixed strategy Nash Equilibrium and obtain a attack probability of about 9%. We used the Nash Equilibrium attacker, with a 9% attack rate, to simulate the effectiveness of even a simple adaptive algorithm in improving the defender's utility.

Our work is aimed at enabling enterprise security teams leverage moving target defense at an architectural level. Organizations running large cloud deployments (50 or more servers) would be the primary beneficiaries of our proposed architecture.

The target audience is also one of the limitations of our work, organizations with smaller could deployments may not be reap the same benefits as a large organization implementing our architecture. Small businesses and independent developers are better served solving low hanging fruits such as vulnerabilities listed under the OWASP Top 10 project [3].

Another limitation of our current work is that we currently rely on simulation data to postulate the results of our work and building a honey pot to collect real world data is one of the future avenues of exploration. Other avenues of exploration include building an improved adaptive algorithm and using machine learning to refine the algorithm in real time.

Thus, while further research is needed to improve the reliability of our architecture, our simulations show that our proposed architecture offers significant security benefits for the defender and under certain scenarios vastly reduces the value for an attacker.

Chapter 8

ACKNOWLEDGEMENTS

This work would not have been possible without the support and advice of Prof. Brent Lagesse who has been a guiding force by sparking my interest in Cyber Security and offering me never ending encouragement to pursue research. I am truly grateful for the time and effort he invested in helping me understand and pursue my research goals.

I am also especially indebted to Prof. Kelvin Sung and Prof. James Fridley, for offering me the opportunity to serve as a Research Assistant in the "College Affordability Model" project for the last two years. The project tested my technical, project management and time management skills and helped significantly improve in all these areas. I would also like to thank Prof. Fridley and Prof. Sung for serving on my committee and for their words of wisdom and advice during times when I've been confused and indecisive. Also, Special thanks to Prof. Munehiro Fukuda for serving on my committee. I would also like to thank Prof. Nancy Kool for her guidance and patience in helping me improve my writing.

I would be remiss not to acknowledge the important role that UWB CSS has played in helping me be a productive student. I would like to thank all the faculty who have invested their time in helping me learn new topics staff. I would also like to thank the administrative staff at UWB with special thanks to Megan Jewell for guiding me through all the administrative processes at UWB from day one. I would also like to thank the UWB ecosystem (Center for International Education, Writing Center, Library, Activities and Recreation Center) for their continued support. Special thanks to Krista Garg and Oz Sener of UWB CIE for guiding me through the complex student

visa processes.

Last but not the least, I thank my friends and family, both in the US and in India for their persistent support with special thanks to my cousin Kartik and his family.

BIBLIOGRAPHY

[1] AWS Security whitepaper. URL: https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Whitepaper.pdf.

[2] Building Trust in a Cloudy Sky. URL: https://www.mcafee.com/us/resources/reports/rp-building-trust-cloudy-sky-summary.pdf.

[3] Category:OWASP Top Ten Project - OWASP. URL: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[4] Completing the Netflix Cloud Migration. URL: https://media.netflix.com/en/company-blog/completing-the-netflix-cloud-migration.

[5] CSD-MTD | Homeland Security. URL: https://www.dhs.gov/science-and-technology/csd-mtd.

[6] Gartner Says Worldwide Public Cloud Services Market to Grow 18 Percent in 2017. URL: http://www.gartner.com/newsroom/id/3616417.

[7] IaaS - Infrastructure as a Service - Gartner IT Glossary. URL: http://www.gartner.com/it-glossary/infrastructure-as-a-service-iaas/.

[8] OS/Linux Distributions using Apache. URL: https://secure1.securityspace.com/s_survey/data/man.201704/apacheos.html.

[9] Shared Responsibility Model - Amazon Web Services (AWS). URL: https://aws.amazon.com/compliance/shared-responsibility-model/.

[10] What Is a Virtual Machine? URL: https://pubs.vmware.com/vsphere-50/topic/com.vmware.vsphere.vm_admin.doc_50/GUID-CEFF6D89-8C19-4143-8C26-4B6D6734D2CB.html.

[11] 30 Most Popular AWS Products - Infographic, May 2016. URL: http://2ndwatch.com/blog/and-the-top-aws-products-of-q1-2016-are/.

[12] Cloud Security Predictions and Trends, December 2016. URL: http://blog.checkpoint.com/2016/12/14/cloud-security-predictions-and-trends/.

[13] Michael Allen. Facsimile | Simulation Library. URL: http://facsim.org/.

[14] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal. Chaos Engineering. *IEEE Software*, 33(3):35–41, May 2016. doi:10.1109/MS.2016.60.

[15] A. Behl. Emerging security challenges in cloud computing: An insight to cloud security challenges and their mitigation. In *2011 World Congress on Information and Communication Technologies*, pages 217–222, December 2011. doi:10.1109/WICT.2011.6141247.

[16] Daniela Brauckhoff, Bernhard Tellenbach, Arno Wagner, Martin May, and Anukool Lakhina. Impact of Packet Sampling on Anomaly Detection Metrics. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 159–164, New York, NY, USA, 2006. ACM. URL: http://doi.acm.org/10.1145/1177080.1177101, doi:10.1145/1177080.1177101.

[17] Ruichuan Chen, Alexey Reznichenko, Paul Francis, and Johanes Gehrke. Towards statistical queries over distributed private user data. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 169–182, 2012. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/chen_ruichuan.

[18] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront. MT6d: A Moving Target IPv6 Defense. In *2011 - MILCOM 2011 Military Communications Conference*, pages 1321–1326, November 2011. doi:10.1109/MILCOM.2011.6127486.

[19] G. Fan, H. Yu, L. Chen, and D. Liu. A Game Theoretic Method to Model and Evaluate Attack-Defense Strategy in Cloud Computing. In *2013 IEEE International Conference on Services Computing*, pages 659–666, June 2013. doi:10.1109/SCC.2013.110.

[20] Diogo AB Fernandes, Liliana FB Soares, Joo V. Gomes, Mrio M. Freire, and Pedro RM Incio. Security issues in cloud environments: a survey. *International Journal of Information Security*, 13(2):113–170, 2014. URL: http://link.springer.com/article/10.1007/s10207-013-0208-7.

[21] Evrim Furuncu and Ibrahim Sogukpinar. Scalable risk assessment method for cloud computing using game theory (CCRAM). *Computer Standards & Interfaces*, 38:44–50, February 2015. URL: http://www.sciencedirect.com/science/article/pii/S0920548914000853, doi:10.1016/j.csi.2014.08.007.

[22] Marc Green, Douglas C. MacFarland, Doran R. Smestad, and Craig A. Shue. Characterizing Network-Based Moving Target Defenses. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, MTD '15, pages 31–35, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2808475.2808484, doi:10.1145/2808475.2808484.

[23] Stephen Groat, Matthew Dunlop, Randy Marchany, and Joseph Tront. Using dynamic addressing for a moving target defense. In *Proceedings of the 6th International Conference on Information Warfare and Security. Academic Conferences Limited*, page 84, 2011. URL: https://books.google.com/books?hl=en&lr=&id=Eobsx_gvPywC&oi=fnd&pg=PA84&dq=using+dynamic+addressing+for+moving+target+defence&ots=eJZD-tpSYX&sig=9sHfAGboWSfC2wMS_qAc4aEbqko.

[24] H. Holm. A Large-Scale Study of the Time Required to Compromise a Computer System. *IEEE Transactions on Dependable and Secure Computing*, 11(1):2–15, January 2014. doi:10.1109/TDSC.2013.21.

[25] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007. doi:10.1109/MCSE.2007.55.

[26] Jafar Haadi H. Jafarian, Ehab Al-Shaer, and Qi Duan. Spatio-temporal Address Mutation for Proactive Cyber Agility Against Sophisticated Attackers. In *Proceedings of the First ACM Workshop on Moving Target Defense*, MTD '14, pages 69–78, New York, NY, USA, 2014. ACM. URL: http://doi.acm.org/10.1145/2663474.2663483, doi:10.1145/2663474.2663483.

[27] Abhinav Jangda, Mohit Mishra, and Bjorn De Sutter. Adaptive Just-In-Time Code Diversification. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, MTD '15, pages 49–53, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2808475.2808487, doi:10.1145/2808475.2808487.

[28] C. A. Kamhoua, L. Kwiat, K. A. Kwiat, J. S. Park, M. Zhao, and M. Rodriguez. Game Theoretic Modeling of Security and Interdependency in a Public Cloud. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 514–521, June 2014. doi:10.1109/CLOUD.2014.75.

[29] P. Kampanakis, H. Perros, and T. Beyene. SDN-based solutions for Moving Target Defense network protection. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6, June 2014. doi:10.1109/WoWMoM.2014.6918979.

[30] Chongkyung Kil, Jinsuk Jun, Christopher Bookholt, Jun Xu, and Peng Ning. Address space layout permutation (ASLP): Towards fine-grained randomization of commodity software. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 339–348. IEEE, 2006. URL: http://ieeexplore.ieee.org/abstract/document/4041179/.

[31] Dmytro Korzhyk, Vincent Conitzer, and Ronald Parr. Security games with multiple attacker resources. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 273, 2011. URL: https://www.cs.duke.edu/~conitzer/multipleIJCAI11.pdf.

[32] Gautam Kumar and Brent Lagesse. Limited Use Cryptographic Tokens in Securing Ephemeral Cloud Servers. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, page 447, Porto, Portugal, February 2017. SCITEPRESS. URL: http://faculty.washington.edu/lagesse/publications/LimitedCryptTokens.pdf.

[33] B. Lagesse and M. Kumar. A Novel Utility and Game-Theoretic Based Security Mechanism for Mobile P2p Systems. In *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 486–491, March 2008. doi:10.1109/PERCOM.2008.49.

[34] Brent Lagesse, Mohan Kumar, and Matthew Wright. AREX: An adaptive system for secure resource access in mobile P2p systems. In *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*, pages 43–52. IEEE, 2008. URL: http://ieeexplore.ieee.org/abstract/document/4627257/.

[35] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981. URL: http://dl.acm.org/citation.cfm?id=358797.

[36] Lawrence M. Lesser. Exploring the Birthday Problem with Spreadsheets. *The Mathematics Teacher*, 92(5):407–411, 1999. URL: http://www.jstor.org/stable/27971021.

[37] Kevin Leyton-Brown and Yoav Shoham. Essentials of game theory: A concise multidisciplinary introduction. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2(1):1–88, 2008. URL: http://www.morganclaypool.com/doi/abs/10.2200/s00108ed1v01y200802aim003.

[38] Douglas C. MacFarland and Craig A. Shue. The SDN Shuffle: Creating a Moving-Target Defense Using Host-based Software-Defined Networking. In *Proceedings*

of the Second ACM Workshop on Moving Target Defense, MTD '15, pages 37–41, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2808475.2808485, doi:10.1145/2808475.2808485.

[39] Mohammad Hossein Manshaei, Quanyan Zhu, Tansu Alpcan, Tamer Bacar, and Jean-Pierre Hubaux. Game theory meets network security and privacy. *ACM Computing Surveys (CSUR)*, 45(3):25, 2013. URL: http://dl.acm.org/citation.cfm?id=2480742.

[40] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Avi Patel, and Muttukrishnan Rajarajan. A survey on security issues and solutions at different layers of Cloud computing. *The Journal of Supercomputing*, 63(2):561–592, 2013. URL: http://link.springer.com/article/10.1007/s11227-012-0831-5.

[41] Klaus Mller. SimPy. URL: https://simpy.readthedocs.io/en/latest/.

[42] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951. URL: http://www.jstor.org/stable/1969529.

[43] A. Panwar, R. Patidar, and V. Koshta. Layered security approach in cloud. In *3rd International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2011)*, pages 214–218, November 2011. doi:10.1049/ic.2011.0083.

[44] Bryan Parno, Jonathan M. McCune, and Adrian Perrig. Bootstrapping trust in commodity computers. In *2010 IEEE Symposium on Security and Privacy*, pages 414–429. IEEE, 2010. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5504802.

[45] N. S. V. Rao, S. W. Poole, F. He, J. Zhuang, C. Y. T. Ma, and D. K. Y. Yau. Cloud computing infrastructure robustness: A game theory approach. In *2012 International Conference on Computing, Networking and Communications (ICNC)*, pages 34–38, January 2012. doi:10.1109/ICCNC.2012.6167441.

[46] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, number 3017 in Lecture Notes in Computer Science, pages 371–388. Springer Berlin Heidelberg, February 2004. DOI: 10.1007/978-3-540-25937-4_24. URL: http://link.springer.com.offcampus.lib.washington.edu/chapter/10.1007/978-3-540-25937-4_24.

[47] Sasha Romanosky. Examining the costs and causes of cyber incidents. *Journal of Cybersecurity*, page tyw001, 2016. URL: http://cybersecurity.oxfordjournals.org/content/early/2016/08/08/cybsec.tyw001.abstract.

[48] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu. A Survey of Game Theory as Applied to Network Security. In *2010 43rd Hawaii International Conference on System Sciences*, pages 1–10, January 2010. doi:10.1109/HICSS.2010.35.

[49] Jyotiprakash Sahoo, Subasish Mohapatra, and Radha Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 222–226. IEEE, 2010. URL: http://ieeexplore.ieee.org/abstract/document/5474503/.

[50] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 298–307. ACM, 2004. URL: http://dl.acm.org/citation.cfm?id=1030124.

[51] Subashini Subashini and Veeraruna Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1):1–11, 2011. URL: http://www.sciencedirect.com/science/article/pii/S1084804510001281.

[52] Ariel Tseitlin. The Antifragile Organization. *Commun. ACM*, 56(8):40–44, August 2013. URL: http://doi.acm.org/10.1145/2492007.2492022, doi:10.1145/2492007.2492022.

[53] Marten Van Dijk, Ari Juels, Alina Oprea, and Ronald L. Rivest. FlipIt: The game of stealthy takeover. *Journal of Cryptology*, 26(4):655–713, 2013. URL: http://link.springer.com/article/10.1007/s00145-012-9134-5.

[54] Maneesh Varshney. SIM.JS | Discrete Event Simulation in JavaScript. URL: http://www.simjs.com/.

[55] Sridhar Venkatesan, Massimiliano Albanese, Kareem Amin, Sushil Jajodia, and Mason Wright. A moving target defense approach to mitigate DDoS attacks against proxy-based architectures. In *IEEE Conference on Communications and Network Security*, 2016. URL: http://amin.kareemx.com/pubs/VenkatesanAlbaneseAminJajodiaWrightCNS2016.pdf.

[56] Kai Wang, Xi Chen, and Yuefei Zhu. Random domain name and address mutation (RDAM) for thwarting reconnaissance attacks. *PloS one*, 12(5):e0177111, 2017. URL: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0177111.

[57] Dave Wichers. OWASP Top-10 2013. *OWASP Foundation, February*, February 2014. URL: https://owasp.org/images/1/17/OWASP_Top-10_2013--AppSec_EU_2013_-_Dave_Wichers.pdf.

[58] S. Yi, D. Kondo, and A. Andrzejak. Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 236–243, July 2010. doi:10.1109/CLOUD.2010.35.

[59] M. Yildiz, J. Abawajy, T. Ercan, and A. Bernoth. A Layered Security Approach for Cloud Computing Infrastructure. In *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, pages 763–767, December 2009. doi:10.1109/I-SPAN.2009.157.

[60] Quanyan Zhu and Tamer Baar. Game-theoretic approach to feedback-driven multi-stage moving target defense. In *International Conference on Decision and Game Theory for Security*, pages 246–263. Springer, 2013. URL: http://link.springer.com/chapter/10.1007/978-3-319-02786-9_15.

[61] Onur zgn and Yaman Barlas. Discrete vs. continuous simulation: When does it matter. In *Proceedings of the 27th international conference of the system dynamics society*, volume 6, pages 1–22, 2009. URL: https://pdfs.semanticscholar.org/d830/b724fd0d16af3dc2281e4f2017b4056fbe38.pdf.

GLOSSARY

attacker

Attacker, within the scope of this thesis refers to any external malicious actor who attempts to discover and exploit vulnerabilities in the defender's cloud systems. 56

AWS

Amazon web services is an on demand IaaS provider with many different cloud computing products on offer, such as EC2 (compute), S3(Storage), RDS (database). 5

CFS

CFS stands for Client Facing Server, which is a server classification we use to represent any server on a cloud system which is exposed to the open Internet. CFS can also be used to denote servers which can potentially be exploited directly by an external malicious actor (see attacker). 13, 14, 20, 24, 30, 35, 42, 43

CTA

CTA stands for Central Trusted Authority, which is a server classification we use to represent any server on a cloud system which is responsible for protecting and maintaining the integrity of sensitive resources. 13, 22, 41, 44

DHT

Distributed Hash Table is a decentralized system which provides a look up table similar to a hash table. DHTs avoid the single point of failure problem present in traditional hash tables. 44

exploitable time

Exploitable time is the amount of time that an attacker has access to a cloud system after the system has been successfully compromised. 26

Hash Chains

Successive application of a cryptographic hash function on a piece of data. see section 2.2.2. 13

hypervisor

Hypervisor in simple terms could be considered as an operating system whose sole purpose is create, run and manage Virtual Machines (VM). Hypervisors are often used in data centers and shared servers. 5, 60

IaaS

Gartner defines "Infrastructure as a service (IaaS) is a standardized, highly automated offering, where compute resources, complemented by storage and networking capabilities are owned and hosted by a service provider and offered to customers on-demand. Customers are able to self-provision this infrastructure, using a Web-based graphical user interface that serves as an IT operations management console for the overall environment. API access to the infrastructure may also be offered as an option." [7]. 1, 3, 9, 11, 56

IPv6

Internet protocol version 6 is the most recent version of the Internet protocol developed by IETF. IPv6 uses 128 bit addressing which allows for 2^128 IP unique addresses. 10

MTD

Moving Target Defense (MTD) is a defensive strategy which aims to increase the uncertainty, complexity, and cost for attackers with relatively low impedance to the defender. 1, 6, 10, 21

Nash Equilibrium

In game theory Nash equilibrium is the state of a game in which no player has anything to gain by changing their strategy. In other words a player cannot gain any extra payoff or utility by changing their strategy. 7, 11, 20, 31, 33

ORM

ORM stands for Object Relational Mapper. An ORM is a library which enables programmers to write requests for data in a programming language of choice instead of using SQL queries. Writing code using an ORM can make a project database agnostic as different databases use different dialects of SQL. ORMs can usually translate between different dialects of SQL. 16

overhead

The additional resources needed to perform a specific task. These resources could be in the form of memory, bandwidth, computation time, or latency. 22

Payoff Matrix

A payoff matrix represents the every player's utility for every state of the world, if the we consider the states of hte world to depend only on the combined actions of the players[37]. 33

Rational player

In game theory a rational player is a player of a game who always acts in a way that maximizes their utility. 33

SDN

Software Defined Networking is a computer networking approach which allows programmatic control of network infrastructure. SDNs are commonly deployed in cloud computing environments or any other large network where flexibility of routing traffic is expected at runtime. 1, 10

sensitive resource

Sensitive resource is a term used to refer to any asset within the cloud infrastructure which is of high value and is worth protecting. Examples of sensitive resources are Database servers and their endpoints, and API Keys. 13

simulation time

Passage of time within in a discrete event simulation is denoted by units of SIM Time. SIM Time, unless otherwise specified, does not translate into real time units. In a discrete event simulation, time transitions between occurrences of two events instantaneously. For example if *event*1 occurs at SIM Time 10 and *event*2 occurs at SIM Time 20, assuming there are not any events occurring between *event*1 and *event*2, the simulation skips the clock to SIM Time 20 without processing times SIM times 11 to 19. iv, 24–27, 30, 31, 36, 37

TTL

TTL stands of Time To Live. The TTL of a CFS is the life time of the server. For example a TTL of 5 hours implies that a particular CFS is alive for 5 hours, after which the CFS is shutdown. 1, 24, 30, 37, 42, 43

Utility

Utility within game theory, is a way to represent the preferences of a player. We assume that rational players would prefer to perform actions which increase or preserve their current utility. Utility is often computed as $Benefit - Cost$. Utility can be considered analogous to Net Profit. 25, 32, 36, 37

VM

"Virtual Machine is a software computer that, like a physical computer is capable of running an operating system." [10]. Virtual Machines are integral to cloud computing. IaaS providers use a hypervisor to share physical hardware resources between many customers. VMs are also used by malware researchers to isolate their operating system environment when testing unknown malware. 9, 43, 57

zero-day

A zero day is a vulnerability in software which has not been previously disclosed. The term originates from the fact that, because the vulnerability was previously undisclosed, the software's author has zero days, or no time to create and distribute patches. 43