Moving Target Defense against DDoS Attacks: An Empirical Game-Theoretic Analysis*

Mason Wright
Computer Science and Engineering
University of Michigan
masondw@umich.edu

Sridhar Venkatesan
Center for Secure Information Systems
George Mason University
svenkate@gmu.edu

Massimiliano Albanese
Center for Secure Information Systems
George Mason University
malbanes@gmu.edu

Michael P. Wellman
Computer Science and Engineering
University of Michigan
wellman@umich.edu

ABSTRACT

Distributed denial-of-service attacks are an increasing problem facing web applications, for which many defense techniques have been proposed, including several moving-target strategies. These strategies typically work by relocating targeted services over time, increasing uncertainty for the attacker, while trying not to disrupt legitimate users or incur excessive costs. Prior work has not shown, however, whether and how a rational defender would choose a moving-target method against an adaptive attacker, and under what conditions. We formulate a denial-of-service scenario as a two-player game, and solve a restricted-strategy version of the game using the methods of empirical game-theoretic analysis. Using agent-based simulation, we evaluate the performance of strategies from prior literature under a variety of attacks and environmental conditions. We find evidence for the strategic stability of various proposed strategies, such as proactive server movement, delayed attack timing, and suspected insider blocking, along with guidelines for when each is likely to be most effective.

Keywords

DDoS, moving target defense, game theory

1. INTRODUCTION

Distributed denial-of-service (DDoS) attacks are a growing real-world problem for web applications, in which an attacker uses a botnet to send a large volume of illegitimate requests to the target application's servers, overwhelming their resources and degrading performance for users [3]. In a conventional defense against DDoS, the defender attempts to filter attack packets but not normal traffic, or to block likely attacker IP addresses [13]. Sophisticated DDoS attacks cannot be stopped by these methods alone, as attackers may use many IP addresses as the source of their packet flood [13], or may structure attack packets to mimic legitimate traffic [5]. As a result, conventional defenses only partially mitigate advanced DDoS attacks, leaving room for improvement through novel methods [22].

Moving Target Defense (MTD), generally speaking, comprises a class of strategies where a defender randomizes its configuration to make disruption more challenging. Through continuous reconfiguration, MTD counters the attacker's ability to gather intelligence and can delay an attack at will. As with any strategy, a defender considering an MTD tactic must weigh its costs against the expected benefits. MTD costs include the cost of moving resources over time and the cost of using suboptimal allocations with some probability (for reduced predictability) [11]. Agent-based simulation provides a way to evaluate and compare MTD techniques, abstracting away some implementation details of the attack and defense [21]. As such, agent-based simulations can help to determine which MTD techniques are rational to use.

Many moving-target techniques have been proposed to help mitigate DDoS attacks [6, 8, 9, 16, 20]. These techniques generally employ an overlay network of servers, which mediates between clients and the target web application. To reduce the impact of DDoS attacks, the defender migrates clients among a large pool of proxy servers, only a small subset of which are active at any point of time. A DDoS attacker can strike the currently active proxies only if it knows which are in use at the time. This moving-target strategy increases the attacker's reconnaissance efforts, at the cost for the defender of running extra servers and migrating users among servers.

Prior work has designed many seemingly reasonable MTD tactics against DDoS, but has not adequately demonstrated that these tactics are rational for a defender to use. Past simulation studies of these policies have assigned the attacker a fixed policy to enact, and in some cases have not considered defender costs when evaluating the utility of MTD strategies. To rigorously evaluate whether an MTD strategy is rational, we must analyze that strategy in a game setting, where the attacker and defender are allowed to select their policies based on the anticipated actions of the opponent [4].

We present a novel game model of a DDoS attack. We

*This work was partially supported by the Army Research Office under grant W911NF-13-1-0421.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MTD'16, October 24 2016, Vienna, Austria

© 2016 ACM. ISBN 978-1-4503-4570-5/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2995272.2995279

formulate the setting as a two-player normal-form game between an attacker and a defender. The two players struggle to influence the quality of service experienced by users of a web application, while keeping their own costs low. The game's design incorporates key tradeoffs for the attacker and defender. For example, the defender seeks to maximize user quality of service while using few proxy servers and performing few client migrations. Similarly, the attacker seeks to minimize user quality of service while using few bots to send attack packets.

We have designed several strategies for each player type: attacker and defender. These strategies are of various types and range from naive to fairly sophisticated. In our game-theoretic analysis, we simulate multiple parameterized forms of each strategy, searching for mixed-strategy Nash equilibria in each game setting.

We employ empirical game-theoretic analysis (EGTA), a simulation-based, Monte Carlo process for finding game-theoretic equilibria in complicated games over restricted strategy spaces [19]. EGTA uses a simulation-based oracle that generates sample payoffs for the agents, for each possible pair of strategies they could play from a finite set. These payoffs induce a normal-form game that can be solved to find Nash equilibria. Recall that a normal-form game is a game with finite player set and action set, in which any mapping of players to actions has a fixed expected payoff for each player. A Nash equilibrium is a strategy profile (assignment of a strategy to each player), such that no agent could increase its expected payoff by changing strategies, while all other agents keep their assigned strategies.

Our overall goal in this study has been to evaluate which MTD policies proposed in prior work are likely to be rational in plausible DDoS settings. We derive insights about under what conditions each MTD tactic is most applicable. By finding strategic equilibria in different experimental conditions via EGTA, we explore in which environments each policy is rational to use. By simulating the equilibrium profiles found by EGTA, we further explore the outcomes that result when equilibrium strategy profiles are enacted.

Our key findings are as follows:

- A DDoS attacker can benefit from directing more attack strength toward servers that appear to have more clients, even if the per-proxy *client count signal* is somewhat noisy.

- A DDoS attacker can benefit from delaying attacks on newly-discovered proxies, to protect the insiders that found the proxies from detection. *Delayed attack timing* is most useful in long games with few total insiders.

- *Proactive server migration* (migrating clients before an attack) is supported as a rational DDoS defense. Proactive migration is especially useful if individual clients often switch proxies, because these migrations can give insiders knowledge of multiple active servers, which could be invalidated by proactive movement.

- The DDoS defender can benefit from blocking clients suspected of being insiders (attackers mimicking legitimate users), even though there is a risk of blocking legitimate users. *Blocking suspected insiders* is most useful when there are few proxies available for the defender to use at a time.

The remainder of the paper is structured as follows. Section 2 discusses related work. Section 3 introduces our DDoS game formulation, the MOTAG Game, whereas Section 4 explains the heuristic strategies our game agents use. Section 5 describes the EGTA methodology. Then, Section 6 introduces our experiments, each of which features a few distinct sets of environment variable settings, and Section 7 presents the results of our experiments. Finally, Section 8 provides some concluding remarks. Additionally, the appendix lists the parameter settings we use for every game environment and attacker or defender policy.

2. BACKGROUND AND RELATE WORK

Several prior works have proposed strategies for DDoS defense that employ a movable overlay network of servers. Some of these works are explicitly within the field of MTD, and some predated it. Keromytis et al. [8] introduced the Secure Overlay Service, a group of secret servers that mediate between clients and a hidden web application. If a secret server is attacked, it can be swapped out for a different server from the pool. Similarly, Khattab et al. [9, 15] proposed "proactive server roaming," a defense system where one server at a time handles client requests, and this server is rotated over a pool of servers at regular intervals. Shi et al. [16] investigated "IP hopping," where the defender moves its server's IP address periodically among a pool of options, following a predefined path. None of these works explicitly uses the term MTD, but all propose moving-target approaches to preventing DDoS attack.

Our work builds upon an MTD model of DDoS attack known as MOTAG, which was introduced by Jia et al. [6, 7] and further analyzed strategically by Venkatesan et al. [17]. In the MOTAG model, clients of a web application are each assigned to a proxy server, which mediates between them and a web application server. The attacker can impersonate normal users to learn the proxyIds of multiple proxy servers, which it can then attack. The defender can migrate clients to different proxy servers, in groups or individually. The defender can also shut down attacked servers and launch new ones from a pool. The attacker seeks to degrade performance for normal users.

An important consideration for the attacker is when to attack a proxy server whose proxyId was recently discovered by an *insider* (malicious user). If the attacker strikes a proxy immediately, the defender may deduce that the client is an insider and block that client from the service. Optimal attack timing was recently investigated by Axelrod and Iliev [1]. They present a model where an attacker's value for successful breach varies over time, and the attacker must decide when to act upon secret knowledge uncovered by surveillance. They find that, intuitively, an attacker should wait longer to exploit its secret information in situations where the breach is likely to be detected immediately.

Our work builds upon prior work on MTD defenses against DDoS, through a novel game model of the MOTAG setting. We adapt the MOTAG environment by endowing it with payoffs (i.e., utilities) for the attacker and defender, an explicit state space, an action space, and transition probabilities, producing a formally defined game. In our game model, the defender will not necessarily use the "greedy shuffling algorithm" proposed by Jia et al. [6], but can use any strategy that maps from the history of its actions and observations to the defender action space. Likewise, the attacker is free

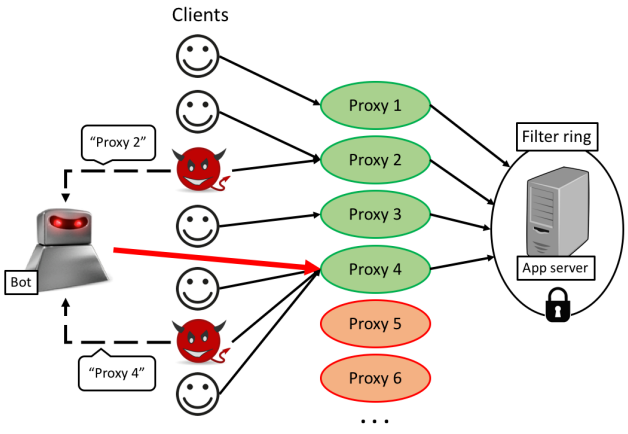

Figure 1: In the MOTAG Game, the defender agent assigns each client to a running proxy (small arrows). The attacker agent can command bots to attack known proxies (larger red arrow). Insiders (devil icons) pose as users and report their proxy's address to the attacker (dashed arrows).

to select the policy that best responds to the defender.

Prior work by Prakash and Wellman [14] showed how EGTA can be applied to study MTD policies for computer security games. Our present work follows this earlier paper's general structure. We define a game model of our strategic setting, solve the game with EGTA under various environmental conditions, and analyze patterns of strategies present in equilibrium for different environments.

The results of our EGTA analysis on the MOTAG Game largely confirm the usefulness of strategies from prior literature. We show settings in which proactive server migration, blocking suspected insiders, or delayed attack timing are used in equilibrium mixed strategies. This indicates that under suitable conditions, a rational agent could plausibly use these techniques. Our detailed results indicate in which environments these strategies are most applicable.

3. GAME FORMULATION

Our *MOTAG Game* formulation is inspired by the MOTAG (moving target) defense scenario that was presented by Jia et al. [6], and expanded upon by Wang et al. [18]. In the MOTAG setting, a vulnerable web application is hosted on an application server that is shielded from direct attack by a filter ring (see Figure 1). Clients can interact with the app server only through a proxy server, of which the defender has several available. A client can connect to a proxy server only if it has been informed of the server's secret address, through logging in to an authentication server.

We propose a two-player game between an attacker that attempts to disrupt the web application, and a defender that seeks to protect the application and its users. Users are non-strategic actors in the environment, who log in and out of the application according to a fixed, predetermined policy.

The attacker commands two types of followers: insiders and bots. An insider behaves like a normal user of the web application, but the attacker can sense the *proxyId* (similar to an IP address) of any server to which an insider is logged in. A bot can launch a flooding attack on a known proxyId, which will reduce the quality of service for users logged in to that proxy. As shown in Figure 1, logged-in insiders (devil icons) inform the attacker of their proxy's address. Bots can launch attacks on any proxy that has been found by an insider (large red arrow).

The defender controls the assignment of clients (users and insiders) to different proxyIds. A proxy will be in one of the following states at any given time: running, starting (will become running in a few time steps), and stopped. The defender can control the state of all the proxies. In Figure 1, running proxies (shown in green) can have clients assigned to them, while stopped proxies (in red) cannot. The defender can also make any client *blocked* (unable to log in or remain logged in) or *not blocked*.

The goal of the defender is to provide a high quality of service for users, at low cost to the defender. The goal of the attacker is to produce low quality of service for users, at low cost to the attacker.

3.1 Environment Parameters

An instance of the MOTAG Game is defined by a set of parameter values. We use parameters to determine simulation length, the number of users, insiders, and bots, and constraints on the power of each agent type—for example, how many proxies the defender can use at once, or the attacker's cost per attacking bot.

Parameter	Meaning
U	normal user count
I	insider count
T	simulation length
p_i^0	initial probability of a client logged-in
p_o	Pr: client logs out if logged-in
p_i	Pr: client attempts to log in if logged-out
p_r	Pr: random client migration on log-in
P	total proxyIds available during run
P'	max running/starting proxies at a time
P_0	initial running proxy count
L	time steps from starting to running proxy
c_p	cost per running/starting proxy per step
c_m	cost per migrated client per step
B'	max attacking bot count at a time
c_b	cost per attacking bot per step
σ	standard deviation in user count signals

Table 1: Environment parameters for a game instance.

In Table 1, we list the complete set of parameters that define a game's environment. Parameters in the top section relate to the entire simulation, in the middle relate mostly to the defender, and at bottom relate to the attacker.

The payoffs of the MOTAG Game are determined by a discrete-event simulation of T time steps. Initially, each of $C = U + I$ clients is logged in with probability p_i^0. The defender has at most P proxies to use overall, of which at most P' can be in use at a time. Each time the defender migrates a client from one proxy to another, this costs c_m. The attacker can use at most B' bots at a time. For each proxy that has an insider logged in, the attacker receives a signal indicating how many total users are logged in, obscured by Gaussian noise with standard deviation σ.

We can characterize some environments as being more or less favorable for the defender, based on their parameter values. The defender benefits from a high user count U and low insider count I. It also benefits from a long simulation length T, which could make it easier to identify and block insiders with many time steps remaining. Each agent naturally benefits from having more resources and lower costs, such as the defender resources P and P', and defender costs c_p and c_m. More subtly, the defender is hurt by a high L, the proxy start-up time. This is because when a proxy is starting, it cannot host clients but still contributes to the defender's cost c_p and resource cap P'. Thus, each starting proxy leads to crowding of users and lower defender payoffs, as well as a greater likelihood that a bot will be attacking a proxy to which users are assigned.

3.2 State Space

The state of the environment progresses by a discrete time process, from time $t = 0$ to time $t = T$. At each time step, the attacker and defender observe parts of the current state and act simultaneously, producing the next state.

At the beginning of a simulation, P_0 proxies are running. All clients are uniformly randomly distributed among these proxies, such that the maximum difference in client count between the proxies is minimized. Each client has an independent Bernoulli probability p_i^0 of being logged in. The logged-in state of each client then progresses as a Markov chain, with transition probabilities p_i and p_o (unless the client is blocked).

The current state of the world can be encoded as a tuple containing the following:

- All environment parameters in Table 1
- $t \in \{0, \ldots, T\}$, the current time step
- A vector of length P, containing for each proxyId:
  - proxy state $\in \{\text{stopped}, \text{starting}, \text{running}\}$
  - steps until running $\in \{\emptyset, 1, \ldots, L\}$
- A vector of length B', containing for each bot:
  - the attacked proxyId $\in \{\emptyset, 1, \ldots, P\}$
- A vector of length U, and another of length I:
  - client state $\in$ { logged in, logged out }
  - client state $\in$ { blocked, not blocked }
  - client's assigned proxyId $\in \{1, \ldots, P\}$
- A set $K \subseteq P$ of proxyIds known to the attacker (initially empty)
- Utility $\in \mathbb{R}$ through time t, for attacker and defender

Note that initially the attacker's set of known proxies, which can be attacked by a bot, is empty. During a time step, any logged-in insider adds its assigned proxyId to K if not present already, thus expanding the attacker's knowledge of potential targets.

3.3 Action Space

At each time step, the attacker directs each of B' bots to attack some known proxy, if desired. The defender selects a new mapping from clients to assigned proxies, starts or stops proxies, and blocks or unblocks clients.

The action of the attacker can be formalized as:

- A vector of length B', specifying for each bot the proxy to attack, $\in \{K \cup \emptyset\}$.

The action of the defender is:

- A vector of length C, containing for each client:
  - the client's new assigned proxyId (which must point to running proxy)
  - the client's new blocked state $\in \{\top, \bot\}$
- A vector of length P, indicating the new state of each proxy (which must respect the start-up time L and maximum proxy constraint P')

3.4 Observations

The MOTAG Game has partial observability to the attacker and defender. We assume, however, that certain aspects of the game are publicly known, including all parameter values in Table 1, the utility function of each agent, and the number of time steps remaining in a run.

It is essential to note that in the MOTAG Game, the defender cannot directly observe which clients are users and which are insiders. If the defender knew a client was an insider, it could act immediately to block that insider, so the insider could not log in and learn any new proxy addresses. The attacker's strategy should generally aim to keep insiders from being identified as such.

It is also critical to note that in the MOTAG Game, the attacker can observe how many users are logged in to each proxy only imperfectly, via a noisy signal (unless $\sigma = 0$). If the attacker knew the user count of each proxy, it could direct more attack units to the more popular proxies and achieve a better payoff.

The attacker is able to observe:

- Known proxy set K
- For each of I insiders:
  - whether the insider is logged in
  - the insider's current observation $\in \{\emptyset, X, 1, \ldots, P\}$, where X indicates the insider tried to log in but was blocked, and $\emptyset$ indicates the insider is not logged in
- For each logged-in insider: a noisy signal of the user count of its proxy, with noise level σ

The defender can observe:

- For each of P proxies:
  - the proxy state $\in$ {stopped, starting, running}
  - steps until running $\in \{\emptyset, 1, \ldots, L\}$
  - count of bots attacking $\in \{0, \ldots, B'\}$
- For each of C clients: whether the client is logged in; whether the client is blocked; and the client's assigned proxyId $\in \{1, \ldots, P\}$

3.5 Payoffs

The attacker and defender in the MOTAG Game struggle to control the quality of service for normal users while keeping their own costs low.

We define quality of service for a logged-in user in a time step, $r_t(u)$, such that it is inversely proportional to the bot count attacking the user's proxy, and decreases linearly with the number of clients sharing that proxy:

$$r_t(u) = \frac{1}{N_B + 1} \times \left(1 - \frac{N_C}{2C}\right),$$

where N_B is the number of bots attacking the proxy, and N_C is the total count of clients logged in to the proxy. A logged-out or blocked user has $r_t(u) = 0$.

We define defender return as the total user quality of service over time, minus the defender costs:

$$R_D = \sum_{t=1}^{T} \left(\sum_{u \in U} r_t(u) \right) - c_p P_t - c_m M_t,$$

where P_t is the number of running or starting proxies at time t, and M_t is the number of users migrated to a different assigned proxyId at t.

Attacker return is defined similarly, taking the negative of quality of service, and accounting for attacker costs:

$$R_A = \sum_{t=1}^{T} - \left(\sum_{u \in U} r_t(u) \right) - c_b B_t,$$

where B_t is the number of attacking bots at time t.

4. HEURISTIC STRATEGIES

An attacker or defender agent in the MOTAG Game may play a strategy that is deterministic or randomized. Strategies may condition the choice of next action on the complete history of an agent's actions and observations, through the current time step t. Our study incorporates several heuristic strategies inspired by prior literature, as well as a set of naive baseline strategies.

4.1 Attacker Strategies

Our most basic naive baseline attacker strategy is the Random Attacker (RA). RA takes two parameters: Bot count b and attack probability a. If K is nonempty, each of b bots will be selected to attack with probability a. Each attacking bot is assigned to a uniform random proxy from K.

A slightly more complex attacker policy is Naive Attacker (NA). NA takes two parameters: Bot count b and maximum proxies to attack p. At each time step, NA attacks the most recently observed $\min(b, p)$ proxies in K. NA attacks each proxy with an approximately equal number of bots B, which is derived as a function of U, I, c_b, and c_p, to be optimal under certain assumptions about defender rationality.

The most sophisticated attacker policy we call Full Attacker (FA). FA takes the same parameters as NA, plus an additional parameter for the probability q of adding a newly observed proxyId to the set that can be attacked. The set of proxies FA may attack, A, is a subset of K; any element of K not in A will be added to A with probability q at each time step. The purpose of waiting to add proxyIds to A with some probability is to prevent the defender from detecting which clients are insiders and blocking the insiders. If the attacker always attacked a proxy immediately after an insider logged in to it, the defender could take advantage of this pattern to identify insiders.

FA adjusts its allocation of bots to proxies based on the signal of user count per proxy (if available). Instead of allocating bots evenly across the most recently observed proxies, FA tends to assign more bots to proxies with higher user counts. To do this efficiently, FA uses a formula relating bot count, estimated user count, and client Count, to assign bots greedily to the most recently observed proxies, until no benefit is expected from assigning another bot.

In environments where a signal of per-proxy user count is present, we include a modified version of FA that ignores this signal. We call this strategy Full Attacker Ignore Signal (FIS). FIS allocates bots evenly across proxies, similarly to NA, regardless of the signal of per-proxy user count. In other respects, FIS is identical to FA. We introduce the FIS strategy to isolate the effect of the attacker's response to per-proxy user count signals, from the effect of other aspects of the FA strategy such as delayed attack timing.

4.2 Defender Strategies

4.2.1 Baseline Defense Policies

We define a baseline Random Defender (RD) policy, which takes parameters for the probability of starting a proxy r_i, the probability of stopping a proxy p_s, and the maximum proxies to use $\bar{p}$. At each time step, RD starts a new proxy with probability r_i, and stops a uniform-randomly selected proxy with probability r_s (subject to the constraints on proxy count). Note that any time a proxy is stopped, its assigned clients must be migrated to some running proxy.

A second baseline policy is the Spread Defender (SD), which has just one parameter, for the maximum proxies to use $\bar{p}$. SD simply gets $\bar{p}$ proxies running at the start of the simulation, migrates all clients to be evenly distributed across these proxies, and does nothing thereafter. SD can perform well in environments where migration costs are high and there are few bots.

Our last baseline policy is the Always Move Defender (AMD), which has one parameter, for the maximum proxies to use $\bar{p}$. AMD always keeps $\bar{p}$ proxies either running or starting. It keeps the clients evenly spread across the greatest number of servers such that all clients can be migrated at every time step, which is $\left\lfloor \frac{\bar{p}}{L+1} \right\rfloor$ (if this number satisfies all constraints). At every time step, all clients are migrated to the proxies that have just entered the running state, and an equal number of new proxies are started. AMD is an effective policy when c_p and c_m are low, but there are many bots and insiders. We use the AMD policy in part as a sanity check, because with realistic settings for the cost parameters, we posit that the AMD policy will not be cost-effective.

4.2.2 No Delay Defender

The No Delay Defender (NDD) operates on the assumption that the attacker will immediately direct some bot to attack any proxyId that an insider encounters for the first time. NDD attempts to determine by experiment which clients are normal users and which are insiders. All insiders will then be blocked by NDD. NDD initially marks all clients as *unknown* type. After any time step in which a proxy is not attacked, all clients who had ever been logged in to that proxy are marked as users and migrated to a proxy having

only users. (This is logical if we assume that any proxy that previously had an insider logged in will be attacked.) Periodically, all unknown-type clients are moved to new proxies and uniformly randomly regrouped across these. If at any time a proxy is attacked for the first time, and exactly one unknown client had been logged in for the previous time step but not in any earlier step, then that client is marked an insider and permanently blocked. Eventually, all insiders will be blocked, and all users will be safe from attack, if the key assumption is correct. NDD should work well against attacker policies that do not delay before attack.

4.2.3 Full Defender

Our most complex defender policy we call Full Defender (FD). FD takes parameters for the threshold score before clients are blocked $\bar{b}$, the probability of proactive movement m, the tendency to adjust proxy count a, and the maximum proxies that can be starting at once s.

FD assigns an insider score $\hat{b}$ to each client, which is initially $\frac{I}{C}$ for all clients. FD tracks which clients had been logged in to each proxy before the proxy was first attacked, considering them to be suspected insiders. When any proxy is attacked for the first time, FD updates $\hat{b}_i$ for each client i that had ever been logged in there, using naive Bayes:

$$\hat{b}_i^{t+1} = \frac{\hat{b}_i^t}{1 - \prod_{j \in Q}(1 - \hat{b}_j^t)},$$

where Q is the set of all clients that had been logged in to the attacked proxy, and $\hat{b}_i^t$ is the estimated probability that client $i \in Q$ is an insider, before the Bayesian update. After each round t of updates, all probabilities $\hat{b}_i, i \in C$ are normalized to sum to I. Once FD has updated the insider probability $\hat{b}$ of all clients, it permanently blocks any client i where $\hat{b}_i \geq \bar{b}$.

When any proxy is attacked, FD attempts to split the clients of that proxy to two new proxies and shut down the attacked one. Subject to expected costs and constraints on the number of running proxies, FD will start two new proxies for each attacked proxy, with the intention of splitting the clients when the new proxies are running. This mechanism is designed to reduce the number of clients affected by an insider by half at each split, eventually isolating an insider completely from users. In order to conserve proxies, proxies that have not been attacked recently are eventually consolidated, reducing c_p costs per time step.

FD can be configured to use a *proactive movement* strategy, migrating clients to new proxies even when no attack has occurred. Proactive movement prevents insiders from maintaining accurate persistent information about the proxyIds that are currently active. This works best in environments where insiders are often moved individually from one proxy to another, as might be necessary for load balancing in a real system. At each time step, with probability m, FD starts a new proxy, for the future assignment of clients from some existing proxy (if allowed by proxy limit constraints). When the proxy is running, it accepts all clients of the proxy that has had the most clients moved away, because if any of those clients is an insider, the proactive movement will cause that insider's knowledge of the old proxy to go stale.

Another key feature of FD is that it attempts to solve analytically for the ideal number of proxies to keep running, based on the number of clients of each type and the cost parameters. A tuning parameter a adjusts how prone FD will be to shut down proxies or start new proxies to approach this ideal proxy count. Based on the tuning parameter, FD may attempt to keep the running proxy count close to its theoretical ideal value, leading to greater defender utility. If the tendency to adjust is too great, however, FD may thrash by constantly starting and stopping proxies, leading to high proxy costs and migration costs, as well as waste from having too many proxies in the starting state.

5. EGTA METHODS

In this work we investigate how rational attacker and defender agents would likely behave in the MOTAG Game, if allowed to play any mixed strategy over the heuristic strategies described above (choosing from a menu of parametrizations). Game-theoretic equilibrium provides a basis for reasoning about which proposed strategies are applicable under different environmental conditions. In order to find approximate strategic equilibria, we employ a technique called empirical game-theoretic analysis (EGTA).

An EGTA study begins with the development of a simulator for the game to be analyzed, which serves as an oracle for sampling from the joint payoff distribution of the agents, given a pure strategy for each agent to play. For each setting of environment parameters, such as the length of the simulation and number of insiders, we run a distinct set of simulations to derive the payoff for each agent.

We begin with a finite set of pure strategies for each agent to select from. In the MOTAG Game, we use 16 attacker strategies and from 9–14 defender strategies per environment. The set of attacker strategies we use is the same for each environment. It contains 2 RA, 3 NA, and 11 FA parametrizations. We use a slightly different set of defender strategies in some environments, depending on what features of the defender policy we want to investigate. In every case, the defender strategy sets include identical choices among 1 RD, 2 SD, 2 AMD, and 1 ND policy. Different strategy sets include 3–8 additional FD parametrizations.

We sample payoffs from many runs of the simulation oracle, for each possible strategy profile in each environment (there are up to $16 \times 14 = 224$ profiles per environment). We take the sample-mean payoff for each agent as an estimate of the expected payoff for the agent of the associated strategy profile. In this study, we sample 2000–4000 runs per strategy profile, per environment. Thus we estimate a two-player normal-form game model, with a complete matrix of expected payoffs, for every environment we study.

Our MOTAG Game scenario is implemented by a discrete-event simulator written in Java. Using the EGTAOnline platform [2], we are able to automate much of the work of running experiments on the University of Michigan's high-performance computing cluster, as well as to store output data automatically in a database.

We use the Gambit software package to solve for approximate mixed-strategy Nash equilibria in each environment [12]. We use the extreme point enumeration solver within Gambit, based on a method introduced by Mangasarian [10].

Once we have found at least one Nash equilibrium for an environment, we run many simulations of each equilibrium mixed strategy profile for that environment. This allows us to estimate the expected values of key outcome variables, such as attacker and defender payoffs, number of clients blocked, and number of proactive server migrations.

Finally, we qualitatively analyze features of the equilib-

ria under different environmental conditions, including both the nature of the mixed strategies that are played, and the expected values of outcomes such as agent payoffs.

6. EXPERIMENTS

We performed four distinct experiments using our MOTAG Game simulator, each designed to investigate a different aspect of the attacker or defender strategy space. The strategic factors we isolate in our experiments are proactive server migration, the use of client count signals by the attacker, blocking suspected insiders, and delayed attack timing. The specific parameter values used for each experiment are laid out in Table 8, in the appendix. Those parameters held in common across all experiments are shown in Table 5, also in the appendix. We list the attacker strategy parametrizations used in all experiments in the Appendix, Table 6. Environments where a signal of per-proxy user count is present (that is, $\sigma < \infty$) include FIS attacker strategies, in addition to the other strategies in Table 6. And the defender strategy sets, which we vary between experiments, are listed in appendix Table 7.

6.1 Client Count Signal Experiment

In our client count signal experiment (CS), we expected that the attacker would obtain a greater expected payoff in the presence of a signal of the per-proxy user count, which it could use to direct more bots to attack proxies with more users. We also expected that the lower the noise of the signal, the more it would increase attacker payoffs. We expected that the benefit of a count signal would be greater in settings where the attacker has many insiders and bots, so that it can capitalize more readily on its knowledge. We also tested the effect of different logged-in client fractions, which either increased over time or are constant.

6.2 Attacker Delay Experiment

Our last experiment concerns attacker delay (AD), or delayed attack timing. We wanted to test whether attackers would be more prone to delay before attacking in simulations of long duration, because the cost of having an insider blocked is greater in a long simulation. The tendency to delay is measured by whether FA is the dominant strategy in equilibrium, and if so, what is the mixed strategy's mean q parameter, indicating how likely an observed proxyId is to be added to the set that may be attacked. For example, if the attacker plays a pure FA strategy with parameter q, the expected delay before a new proxyId will be attacked is $\frac{1}{q}$. We expected that the attacker delay would be greater in settings with lower attacker power, meaning fewer insiders I, more proxies allowed to run at once P', and lower cost per proxy c_p. This is because when there are few insiders, for example, the attacker must be more careful not to let an insider be blocked.

6.3 Proactive Migration Experiment

In the proactive migration experiment (PM), we expected that proactive server migration would be heavily used by the defender in settings where individual clients are sometimes migrated to a different proxy by Nature, but not in settings where this does not occur. To test this, we introduced an environment parameter for random migration, p_r, such that any time a client logs in, with probability p_r that client will be migrated to a uniform-random running proxy. This is designed to simulate a setting where for load balancing or other reasons, clients cannot always remain mapped to a proxy as desired by the defender. We also tested the effects on proactive migration of high or low per-proxy cost c_p, and of a logged-in client fraction that is either increasing or constant.

6.4 Insider Blocking Experiment

The insider blocking experiment (IB) analyzes under what conditions the defender has a greater tendency to block suspected insiders, as evidenced either by a low blocking threshold $\bar{b}$ at equilibrium, or by a large number of insiders blocked in simulations of equilibrium strategy profiles. We expected that the defender would be more prone to block suspected insiders in settings with random client migrations ($p_r > 0$), because when clients are randomly moved about as individuals, this allows some insiders to learn the proxyIds of proxies to which they can no longer log in; this information will be made useless if the old proxy's users are migrated to a different proxy. We also expected that more blocking would occur in settings with fewer insiders, because in a world with many insiders, there is little gained by blocking a few, and blocking too many would lead to an unacceptable number of users being blocked by mistake. We also varied the number of proxies that can run at a time P' and per-proxy cost c_p.

7. RESULTS

We present results from our four experiments (groups of parametrized environments) in turn, where each experiment analyzes the effects of a particular environment element or strategy. Our results pertain to the approximate Nash equilibrium strategy profiles we found for each game environment, which represent actions a rational attacker and defender might take. We draw conclusions from which strategies are used with positive probability at equilibrium, and which are weighted most heavily. These strategies' presence in equilibria indicates that they are rational to use in a particular environment. We also draw conclusions from the outcomes that result when equilibrium strategies are enacted, by sampling from the Nash equilibrium mixed strategies and running our simulator. Key outcomes include the payoff to attacker and defender agents, as well as how often the agents perform actions such as blocking a suspected insider.

7.1 Client Count Signal Results

Our results indicate that, as one would expect, attacker payoffs at equilibrium tend to be greater in settings where the observable signal of per-proxy user count has lower noise. With a low-noise per-proxy user count signal, an attacker can decrease users' quality of service by directing more bots to attack proxies that host more users. As shown in Table 2, for each insider count tested (3, 5, 12, or 24 insiders), there is an increasing trend in attacker mean payoff, with decreasing noise in the per-proxy user count signal.

However, there is no significant difference in attacker payoff between conditions of no signal ($\sigma = \infty$) and extremely noisy signal ($\sigma = 10$), likely because the Full Attacker strategy overfits to noise when it allocates its bots. If the per-proxy user count signal is sufficiently noisy, it is almost worthless to the attacker. Moreover, when the attacker has very few insiders and bots are costly ($I = 3$ and $c_b = 2.0$), there is no significant increase in attacker payoff with increasing signal quality. This is expected, as the attacker

can detect user counts only through its insiders, and it must allocate multiple bots to take full advantage of user count information.

I	c_b	σ	R_A
24	0.05	∞	–10484
24	0.05	10	–10481
24	0.05	0	–10259
12	0.1	∞	–13004
12	0.1	2	–12856
12	0.1	0	–12830
5	0.5	∞	–13034
5	0.5	2	–12999
5	0.5	0	–12750
3	2.0	∞	–14491
3	2.0	10	–14488
3	2.0	0	–14488

Table 2: Mean attacker payoff R_A, for environments that vary by number of insiders I, cost per bot c_b, and standard deviation σ of noise in the signal of per-proxy user count.

Observe that the noise level σ has less impact on attacker payoffs, in settings where the attacker has fewer insiders. In settings with many insiders I and low per-bot cost c_b, there is a larger difference in attacker payoff between the cases of no signal ($\sigma = \infty$) and noiseless signal ($\sigma = 0$).

In environments where a signal is present ($\sigma \neq \infty$), we allow the attacker to use several parametrizations of a Full Attacker Ignore Signal (FIS) strategy, in addition to the Full Attacker (FA) strategy. FIS is identical to FA, except that it allocates bots evenly across the most recently observed proxies, even if a per-proxy user count signal is present. To further evaluate whether a user count signal is helpful to the attacker, we can test with what probability the equilibrium attacker mixed strategy uses that signal in each setting. If the attacker elects to use FIS instead of FA at equilibrium, that will indicate that the user count signal is not helpful.

We find that in settings with very noisy signals of per-proxy user count ($\sigma = 10$), the equilibrium attacker does not use the FA strategy. This might be because with such noisy signals, FA tends to allocate bots unwisely across proxies, compared to a balanced allocation, with an equal number of bots attacking each proxy. In settings with low signal noise ($\sigma = 2$), the attacker uses the FA strategy with high probability (1 or 0.965). And in settings with noiseless signal ($\sigma = 0$), the attacker also uses FA with high probability (0.904, 1, 1), except in the case where $I = 3$ and $c_b = 2.0$, where FA is not used. Perhaps when the insider count is very low and bot cost is high, there is no advantage to attempting to allocate bots based on server load, and it is better simply to assign one bot each to the most likely-active proxies.

7.2 Attacker Delay Results

We want to evaluate whether the attacker has a greater tendency to delay its attacks on newly-discovered proxies in certain environments, such as where there are few insiders and a long simulation length. To this end, we define a measure, $\bar{D}$, which is the expected delay before the equilibrium attacker mixed strategy will add a newly-discovered proxy to the set that may be attacked. For example, if the attacker plays FA, $q = 0.2$ with probability $\frac{1}{2}$ and FA, $q = 1.0$ with probability $\frac{1}{2}$, the expected delay before a potential attack is $\frac{1}{2} \times \frac{1}{0.2} + \frac{1}{2} \times \frac{1}{1} = 3$. We let the expected delay for any strategy other than FA and FIS be 1, signifying no extra delay. Thus, $\bar{D}$ represents how many time steps we might expect the attacker to delay before striking a newly found proxy. A greater value of $\bar{D}$ indicates a more cautious attacker and might be expected in settings with fewer insiders or a longer simulation. A cautious attacker delays striking newly discovered proxies, in spite of the loss from temporarily allowing their users to obtain high quality of service, in order to protect insiders from detection and blocking.

I	p_r	T	$\bar{D}$
5	0.3	1000	20.000
5	0.3	300	3.333
5	0.0	1000	1.110
5	0.0	300	1.000
12	0.3	1000	3.333
12	0.3	300	1.000
12	0.0	1000	1.090
12	0.0	300	1.072
24	0.3	1000	1.000
24	0.0	300	1.000

Table 3: Expected delay before equilibrium attacker mixed strategy adds a new proxy to the set that may be attacked. Larger values of $\bar{D}$ indicate a more cautious attacker.

As shown in Table 3, the equilibrium attacker uses an extra delay before attacking new proxies if there are few insiders ($I < 12$) and there is some random migration of clients ($p_r > 0$). This demonstrates that it in certain environments, it is reasonable for the attacker to protect the secrecy of its insiders, even at the cost of not using their information immediately.

The attacker's expected delay is longer in settings with fewer insiders, on condition that there is some random migration. With 24 insiders, we do not observe any delay at all in equilibrium ($\bar{D} = 1$). This is likely because with so many insiders, there is little cost to the attacker if several are blocked by the defender. With 12 insiders, there is significant delay only with a long simulation and some random migrations. With 5 insiders, however, the attacker delays attack even in a shorter simulation of 300 time steps, if there is some random migration. The attacker achieves a greater payoff by protecting those few insiders from discovery and blocking than by attacking as soon as the insiders learn about new proxies.

The attacker delays attack more in long simulations than in short ones, possibly because in a long simulation, it is more harmful if many insiders are caught and blocked early. With 5 insiders and $p_r = 0.3$, the attacker delays by a mean of 20 time steps in a long simulation of 1000 time steps, but only 3.333 time steps in a shorter simulation of 300. Similarly, with 12 insiders and $p_r = 0.3$, the attacker delays by 3.333 in the longer simulation, but not at all ($\bar{D} = 1$) in a shorter one.

7.3 Proactive Migration Results

We found the equilibrium defender uses proactive move-

ment strategies with positive probability in several environments. This indicates that proactive movement is sometimes useful, even though proactive movement requires the defender to start a new proxy which cannot accept clients for L time steps, and to pay the cost of migrating clients. Proactive movement is not used at equilibrium in environments with high proxy cost, $c_p = 2.0$, however. This is likely because in our model, if the per-proxy cost parameter is sufficiently high, it is not cost-effective for the defender to start a new proxy except when the defender is reacting to an attack.

p_r	p_i	p_o	c_p	$\bar{M}$
0.3	0.006	0.002	0.5	0.200
0.0	0.006	0.002	0.5	0.034
0.3	0.004	0.004	0.5	0.500
0.0	0.004	0.004	0.5	0.000
0.3	0.05	0.05	0.5	0.500
0.0	0.05	0.05	0.5	0.048

Table 4: Equilibrium tendency $\bar{M}$ for defender to perform proactive movement in a given time step. Results are shown for each environment in experiment PM where $c_p = 0.5$. With greater c_p, proactive movement is not used at equilibrium.

The equilibrium defender has a greater tendency to perform proactive movement in environments with positive random migration probability—$p_r > 0$. Recall that in these settings, any time a client logs in to a proxy server, with probability p_r it must be reassigned to a uniform random server, to simulate load balancing or other constraints. As shown in Table 4, in higher-p_r settings the equilibrium defender has greater mean probability of proactive movement. In this table we present $\bar{M}$, the mean over defender policies in the equilibrium mixed strategy, of the FD proactive movement parameter m (or 0 if not playing FD). An FD defender will proactively migrate one proxy in each time step with probability m, if its resource constraints permit.

This result is reasonable and expected. When p_r is high, some insiders have likely been migrated individually from proxies that still host users. If users at the former proxy of such an insider are proactively migrated to a new proxy, then the insider's knowledge of the old proxy is invalidated. That is, if the attacker strikes the former proxy of the insider, no users will be affected, because the users will have been moved to a new proxy the attacker does not know of.

Conversely, when $p_r = 0$, the defender will migrate all clients of a proxy at once when migrations are performed. This policy makes an insider's knowledge of its former proxies useless, because those proxies will be empty of users. Thus, the defender cannot benefit from proactive movement.

7.4 Insider Blocking Results

We find that under certain conditions, a rational defender will block clients suspected of being insiders, in spite of the risk of blocking legitimate users and dropping their quality of service to zero. In all of the equilibria we find for the nine environments in experiment IB, the defender blocks suspected insiders with positive probability.

In a given equilibrium, we can measure the tendency of the defender mixed strategy to block suspected insiders as follows. A defender playing the FD policy has a parameter $\bar{b}$, such that the defender will block any client it believes has a probability greater than $\bar{b}$ of being an insider. We can take the weighted average of $\bar{b}$ over the defender's mixed strategy, assigning a value of $\bar{b} = 1$ to any non-FD strategy (i.e., no blocking). We thus derive a measure we denote b^*, the mean blocking threshold of the equilibrium mixed strategy. Lower values of b^* indicate a greater tendency to block suspected insiders, because a lower probability of being an insider is sufficient for blocking.

We find that across the six environments in experiment IB where the insider count is held constant at $I = 5$, the defender is consistently more prone to blocking clients when there are fewer proxies available P'. In settings with only 10 proxies available at a time, b^* is in $(0.814, 0.840, 0.747)$; but with 20 proxies available, b^* is higher, in $(0.9, 0.9, 0.901)$. This result is sensible, because each insider has the potential to cause greater harm to users when there are fewer proxies and each proxy has more users on average. In addition, with less space available for shuffling clients among different proxies, it may be more difficult to achieve a high level of belief that a client is an insider.

We did not find any clear trend in the tendency to block suspected insiders with respect to other factors of the environment. For example, we did not find evidence of an effect of insider count I or random move frequency p_r.

8. DISCUSSION

Through EGTA, we have demonstrated the usefulness of four strategies related to moving target defense against DDoS attack. We formalized the MOTAG setting from prior work as the MOTAG Game, with clearly defined payoffs for attacker and defender, along with action spaces and an observation function. We developed a discrete event simulator of the MOTAG Game, which serves as a generative model for payoffs in this environment, conditioned on attacker and defender policies. EGTA allows us to find approximate Nash equilibria of various parametrized environments, helping us explore under what conditions a strategy is rational to use.

Our investigation provides evidence for the effectiveness of the strategies tested, under suitable conditions.

- An attacker can benefit from directing more attack power to proxies that appear to hold more users, as long as the signal of per-proxy user count is not too noisy. If the attacker lacks sufficient bots to attack many proxies at a time, however, it does not benefit from a signal of per-proxy user counts.
- The attacker can gain by delaying its attack on newly discovered proxies, in settings where clients must sometimes be migrated individually. This is especially so in long simulations, if there are few insiders, because the attacker has more to lose if a few insiders are blocked early in the simulation.
- Proactive movement of clients is useful for the defender, especially if individual clients must be migrated at times, and if the cost per proxy is not too high.
- Blocking suspected insiders helps the defender, if there are only a few insiders present. If there are many insiders, the defender gains little from attempting to block them, because it is impossible to block most insiders without a high risk of blocking normal users.

Findings on the relative benefits of alternative strategies are sensitive to changes in a game's parameters. For example, we find that if the cost per proxy is increased drastically, equilibrium defenders tend to adopt the simplistic Spread Defender policy, which uses few proxies and performs minimal client migrations. In another example, if the number of insiders is extremely high, the attacker never uses any delay before attacking a newly discovered proxy, presumably because it can afford to have many of its insiders blocked.

Like any EGTA study, this work should be interpreted with the understanding that only a small, finite set of strategies was sampled in each environment. In this study, we explored several different strategy families, including multiple parametrizations of most, in an attempt to provide agents with multiple effective policies for every environment. Nonetheless, it is possible that some other policy performs better than these, and that rational agents would produce qualitatively different results from what we have shown. We consider only a few environments per experiment, but possibly in other environments the trends we observe would not continue. In spite of these caveats, we believe our work gives encouraging support for the claim that our proposed strategies are effective, in reasonable environment settings.

In future work, the MOTAG Game could be extended in many ways. A minor extension could explore new strategies, comparing the equilibria found by EGTA in the enlarged strategy space to the equilibria we presented here. Similarly, future work could investigate the effects of a modified payoff function or new environment settings.

A more ambitious extension to the MOTAG Game could allow the defender alternative forms of DDoS defense, besides the proxy server layer from MOTAG. This extension would redefine the MOTAG Game as a general DDoS game, in which the MOTAG defense could be represented, as well as fundamentally different defense methods.

9. REFERENCES

[1] R. Axelrod and R. Iliev. Timing of cyber conflict. *Proceedings of the National Academy of Sciences*, 111(4):1298–1303, 2014.

[2] B.-A. Cassell and M. P. Wellman. EGTAOnline: An experiment manager for simulation-based game studies. In *Multi-Agent-Based Simulation XIII*, pages 85–100. 2012.

[3] R. K. C. Chang. Defending against flooding-based distributed denial-of-service attacks: a tutorial. *IEEE Communications Magazine*, 40(10):42–51, 2002.

[4] G. Cybenko, S. Jajodia, M. P. Wellman, and P. Liu. Adversarial and uncertain reasoning for adaptive cyber defense: Building the scientific foundation. In *Information Systems Security*, pages 1–8. 2014.

[5] B. Gupta, R. C. Joshi, and M. Misra. Distributed denial of service prevention techniques. *International Journal of Computer and Electrical Engineering*, 2(2):268–276, 2012.

[6] Q. Jia, K. Sun, and A. Stavrou. MOTAG: Moving target defense against internet denial of service attacks. In *22nd International Conference on Computer Communications and Networks*, pages 1–9, 2013.

[7] Q. Jia, H. Wang, D. Fleck, F. Li, A. Stavrou, and W. Powell. Catch me if you can: a cloud-enabled DDoS defense. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 264–275, 2014.

[8] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. *ACM SIGCOMM Computer Communication Review*, 32(4):61–72, 2002.

[9] S. M. Khattab, C. Sangpachatanaruk, R. Melhem, D. Mossé, and T. Znati. Proactive server roaming for mitigating denial-of-service attacks. In *International Conference on Information Technology: Research and Education*, pages 286–290, 2003.

[10] O. L. Mangasarian. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(4):778–780, 1964.

[11] P. McDaniel, T. Jaeger, T. F. La Porta, N. Papernot, R. J. Walls, A. Kott, L. Marvel, A. Swami, P. Mohapatra, S. V. Krishnamurthy, et al. Security and science of agility. In *First ACM Workshop on Moving Target Defense*, pages 13–19, 2014.

[12] R. D. McKelvey, A. M. McLennan, and T. L. Turocy. Gambit: Software tools for game theory. 2006.

[13] J. Mirkovic and P. Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[14] A. Prakash and M. P. Wellman. Empirical game-theoretic analysis for moving target defense. In *Second ACM Workshop on Moving Target Defense*, pages 57–65, 2015.

[15] C. Sangpachatanaruk, S. M. Khattab, T. Znati, R. Melhem, and D. Mossé. Design and analysis of a replicated elusive server scheme for mitigating denial of service attacks. *Journal of Systems and Software*, 73(1):15–29, 2004.

[16] L. Shi, C. Jia, S. Lü, and Z. Liu. Port and address hopping for active cyber-defense. In *Intelligence and Security Informatics*, pages 295–300. 2007.

[17] S. Venkatesan, M. Albanese, K. Amin, S. Jajodia, and M. Wright. A moving target defense approach to mitigate DDoS attacks against proxy-based architectures. In *IEEE Conference on Communications and Network Security*, 2016.

[18] H. Wang, Q. Jia, D. Fleck, W. Powell, F. Li, and A. Stavrou. A moving target DDoS defense mechanism. *Computer Communications*, 46:10–21, 2014.

[19] M. P. Wellman. Methods for empirical game-theoretic analysis (extended abstract). In *21st National Conference on Artificial Intelligence*, 2006.

[20] P. Wood, C. Gutierrez, and S. Bagchi. Denial of Service Elusion (DoSE): Keeping clients connected for less. In *34th Symposium on Reliable Distributed Systems*, pages 94–103, 2015.

[21] J. Xu, P. Guo, M. Zhao, R. F. Erbacher, M. Zhu, and P. Liu. Comparing different moving target defense techniques. In *First ACM Workshop on Moving Target Defense*, pages 97–107, 2014.

[22] S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys & Tutorials*, 15(4):2046–2069, 2013.

APPENDIX

A. EXPERIMENTAL CONDITIONS

In this appendix, we provide detailed information about the parameter settings we use for every game environment and attacker or defender policy. Table 5 lists the environment parameter settings that are common across all experiments. Table 6 lists the attacker strategy parametrizations used in all experiments, whereas the defender strategy sets, which we vary between experiments, are listed in Table 7. Finally, the specific parameter values used for each experiment are shown in Table 8.

U	P	P_0	L	c_m
100	1000	$\frac{P'}{2}$	10	0.01

Table 5: Environment parameter settings that are common across all experiments.

Strategy	b	p	q	a	Environments
FA	8	8	0.03	-	$\mathbb{U}$
FA	8	8	0.2	-	$\mathbb{U}$
FA	8	8	1.0	-	$\mathbb{U}$
FA	20	10	0.05	-	$\mathbb{U}$
FA	20	10	0.3	-	$\mathbb{U}$
FA	40	10	0.05	-	$\mathbb{U}$
FA	40	10	0.3	-	$\mathbb{U}$
FA	40	10	1.0	-	$\mathbb{U}$
FA	40	20	0.03	-	$\mathbb{U}$
FA	40	20	0.3	-	$\mathbb{U}$
FA	40	20	1.0	-	$\mathbb{U}$
NA	10	10	-	-	$\mathbb{U}$
NA	40	10	-	-	$\mathbb{U}$
NA	40	20	-	-	$\mathbb{U}$
RA	10	-	-	1.0	$\mathbb{U}$
RA	40	-	-	0.5	$\mathbb{U}$
FIS	8	8	0.03	-	$\sigma < \infty$
FIS	20	10	0.05	-	$\sigma < \infty$
FIS	40	10	0.3	-	$\sigma < \infty$
FIS	40	10	1.0	-	$\sigma < \infty$
FIS	40	20	1.0	-	$\sigma < \infty$

Table 6: Attacker strategy set. The FA, NA, and RA attacker strategies are used in every environment. The FIS strategies are also used in those CS environments where a signal of per-proxy user counts is present ($\sigma < \infty$).

Strategy	$\bar{p}$	$\bar{b}$	m	a	s	r_i	r_s	Experiments
RD	20	-	-	-	-	0.05	0.05	CS, AD, PM, IB
SD	10	-	-	-	-	-	-	CS, AD, PM, IB
SD	20	-	-	-	-	-	-	CS, AD, PM, IB
AMD	10	-	-	-	-	-	-	CS, AD, PM, IB
AMD	20	-	-	-	-	-	-	CS, AD, PM, IB
NDD	20	-	-	-	-	-	-	CS, AD, PM, IB
FD	20	0.95	0.0	0.1	4	-	-	CS, AD, PM, IB
FD	20	0.7	0.0	0.1	4	-	-	CS, AD
FD	20	0.85	0.0	0.1	4	-	-	CS, AD
FD	20	0.9	0.0	0.05	6	-	-	CS, AD
FD	20	1.0	0.0	0.1	4	-	-	CS, AD
FD	20	0.95	0.05	0.1	4	-	-	CS, PM
FD	20	0.95	0.1	0.1	4	-	-	CS, PM
FD	20	0.9	0.0	0.0	6	-	-	CS
FD	20	0.95	0.05	0.3	4	-	-	PM
FD	20	0.95	0.2	0.1	4	-	-	PM
FD	20	0.95	0.5	0.1	4	-	-	PM
FD	20	0.95	0.0	0.3	4	-	-	IB
FD	20	0.95	0.0	0.6	4	-	-	IB

Table 7: Defender strategy set. Each experiment uses a different subset of these strategies. The Experiments column lists all the experiments that use each strategy.

Experiment	p_i^0	p_i	p_o	p_r	c_p	σ	I	B'	P'	T	c_b
CS	0.25	0.006	0.002	0.0	0.5	∞	24	40	20	300	0.05
CS	0.25	0.006	0.002	0.0	0.5	10.0	24	40	20	300	0.05
CS	0.25	0.006	0.002	0.0	0.5	0.0	24	40	20	300	0.05
CS	0.25	0.006	0.002	0.0	0.5	∞	12	40	20	300	0.1
CS	0.25	0.006	0.002	0.0	0.5	2.0	12	40	20	300	0.1
CS	0.25	0.006	0.002	0.0	0.5	0.0	12	40	20	300	0.1
CS	0.5	0.005	0.005	0.0	0.5	∞	5	10	20	300	0.5
CS	0.5	0.005	0.005	0.0	0.5	2.0	5	10	20	300	0.5
CS	0.5	0.005	0.005	0.0	0.5	0.0	5	10	20	300	0.5
CS	0.5	0.005	0.005	0.0	0.5	∞	3	10	20	300	2.0
CS	0.5	0.005	0.005	0.0	0.5	10.0	3	10	20	300	2.0
CS	0.5	0.005	0.005	0.0	0.5	0.0	3	10	20	300	2.0
AD	0.5	0.01	0.01	0.3	0.5	∞	5	40	20	1000	0.1
AD	0.5	0.01	0.01	0.3	0.5	∞	5	40	20	300	0.1
AD	0.5	0.01	0.01	0.0	0.5	∞	5	40	20	1000	0.1
AD	0.5	0.01	0.01	0.0	0.5	∞	5	40	20	300	0.1
AD	0.5	0.01	0.01	0.3	2.0	∞	12	40	10	1000	0.1
AD	0.5	0.01	0.01	0.3	2.0	∞	12	40	10	300	0.1
AD	0.5	0.01	0.01	0.0	2.0	∞	12	40	10	1000	0.1
AD	0.5	0.01	0.01	0.0	2.0	∞	12	40	10	300	0.1
AD	0.5	0.01	0.01	0.3	4.0	∞	24	40	10	1000	0.1
AD	0.5	0.01	0.01	0.0	4.0	∞	24	40	10	300	0.1
PM	0.25	0.006	0.002	0.3	0.5	∞	5	40	20	300	0.1
PM	0.25	0.006	0.002	0.0	0.5	∞	5	40	20	300	0.1
PM	0.5	0.004	0.004	0.3	0.5	∞	5	40	20	300	0.1
PM	0.5	0.004	0.004	0.0	0.5	∞	5	40	20	300	0.1
PM	0.5	0.05	0.05	0.3	0.5	∞	5	40	20	300	0.1
PM	0.5	0.05	0.05	0.0	0.5	∞	5	40	20	300	0.1
PM	0.25	0.006	0.002	0.3	2.0	∞	5	40	20	300	0.1
PM	0.25	0.006	0.002	0.0	2.0	∞	5	40	20	300	0.1
PM	0.5	0.004	0.004	0.3	2.0	∞	5	40	20	300	0.1
PM	0.5	0.004	0.004	0.0	2.0	∞	5	40	20	300	0.1
IB	0.5	0.05	0.05	0.9	2.0	∞	5	40	10	300	0.1
IB	0.5	0.05	0.05	0.3	2.0	∞	5	40	10	300	0.1
IB	0.5	0.05	0.05	0.0	2.0	∞	5	40	10	300	0.1
IB	0.5	0.05	0.05	0.9	0.5	∞	5	40	20	300	0.1
IB	0.5	0.05	0.05	0.3	0.5	∞	5	40	20	300	0.1
IB	0.5	0.05	0.05	0.0	0.5	∞	5	40	20	300	0.1
IB	0.5	0.05	0.05	0.9	0.5	∞	12	40	20	300	0.1
IB	0.5	0.05	0.05	0.3	0.5	∞	12	40	20	300	0.1
IB	0.5	0.05	0.05	0.0	0.5	∞	12	40	20	300	0.1

Table 8: Environment parameter settings for each experiment.