Hindawi
Security and Communication Networks
Volume 2020, Article ID 8873258, 12 pages
https://doi.org/10.1155/2020/8873258

Research Article

A Client Bootstrapping Protocol for DoS Attack Mitigation on Entry Point Services in the Cloud

Hussain M. J. Almohri,[1] **Mohammad Almutawa**,[1] **Mahmoud Alawadh**,[1] **and Karim Elish**[2]

[1]*Department of Computer Science, Kuwait University, Kuwait City, Kuwait*
[2]*Department of Computer Science, Florida Polytechnic University, Lakeland, USA*

Correspondence should be addressed to Hussain M. J. Almohri; almohri@cs.ku.edu.kw

Received 13 May 2020; Revised 25 June 2020; Accepted 4 July 2020; Published 23 July 2020

Academic Editor: Mamoun Alazab

This paper presents a client bootstrapping protocol for proxy-based moving target defense system for the cloud. The protocol establishes the identity of prospective clients who intend to connect to web services behind obscure proxy servers in a cloud-based network. In client bootstrapping, a set of initial line of defense services receive new client requests, execute an algorithm to assign them to a proxy server, and reply back with the address of the chosen proxy server. The bootstrapping protocol only reveals one proxy address to each client, maintaining the obscurity of the addresses for other proxy servers. Hiding the addresses of proxy servers aims to lower the likelihood that a proxy server becomes the victim of a denial-of-service (DoS) attack. Existing works address this problem by requiring the solution of computationally intensive puzzles from prospective clients. This solution slows the progression of attacks as well as new clients. This paper presents an alternative idea by observing that limited capacity of handling initial network requests is the primary cause of denial-of-service attacks. Thus, the suggested alternative is to utilize cost-effective high-capacity networks to handle client bootstrapping, thus thwarting attacks on the initial line of defense. The prototype implementation of the protocol using Google's firebase demonstrates the proof of concept for web services that receive network requests from clients on mobile devices.

1. Introduction

Denial-of-service (DoS) attacks on web services in cloud-based virtual networks continue to threaten small or medium-sized networks by exhausting the available memory and computation power of the hosting machines. Small or medium-sized networks are particularly vulnerable because of the budget limitation, severely restricting the computation capacity of the machines that serve external clients. Thus, attackers can win the resource race against the target network by using simple denial-of-service vulnerabilities such as the ones in prominent web server software. A growing interest is in the use of moving target defense (MTD) strategy against DoS attacks on vulnerable networks. In this case, target services dynamically change IP addresses and introduce diversity in the hosting machines to gain the advantage of time. The moving target defense model benefits from the elasticity of the cloud, which allows for swift and dynamic responses to attacks, using programmable firewall rules and network interfaces applied to elastic computing resources.

Moving target defense provides a strong defense strategy against DoS attacks without focusing on the details of specific network service vulnerabilities that attackers exploit. The central idea is to increase the DoS attacker's effort in finding and attacking large services in the network. Formally, suppose that a function search $(\mathcal{S}, N)$ is used by the denial-of-service attacker, given a possible search space $\mathcal{S}$ to find the target addresses in a network N. With no moving target defense, assume that search $(\mathcal{S}, N)$ terminates with the complexity $\mathcal{O}(f(n))$, for a typically linear $f(n)$ number of possible addresses (usually requiring a Nmap scan [1]). The key to maximizing the attacker's search effort is in randomizing the network address search space. Thus,

moving target defense aims to maximize $\mathcal{O}(f(n))$, resulting in a higher-order polynomial or ideally an exponential $f(n)$. State-of-the-art moving target defense systems use proxy servers that mediate clients and application servers. The IP addresses of the proxy servers are only known to *bootstrapped* clients that have been through identification, authentication, and assignment to specific proxy servers. Attacker clients (without knowing the proxy IP addresses) should search through an IP address space to find the target proxy servers. That said, the network should allow new clients to connect to the proxy servers by first interacting with machines in an initial line of defense. The initial interaction would perform client bootstrapping and allow future communication with the proxy servers. The problem is that machines that perform the initial interaction with unknown clients are themselves susceptible to denial-of-service attacks. A weak initial client bootstrapping would defeat the purpose of moving target defense.

1.1. Problem Statement. Scalable and effective client bootstrapping for moving target defense is the focus of this work. The bootstrapping process requires a set of entry point servers as the first line of defense, referred to as the initial point, that is only responsible for bootstrapping clients. Since proxy server IP addresses are not public, a prospective client informs the initial point about the intent to use the hosted services. The initial point executes a bootstrapping protocol to identify, authenticate, and register the prospective client. If successful, this protocol terminates with informing the client about a secret address of a proxy $p \in R$ of the set of currently executing proxy servers R. Using this information, the client can now connect to p, which relays the client requests and the application server's responses. Client bootstrapping is a simple but crucial process that requires special considerations. Without protecting the machines in the initial point, the entire moving target defense is useless. The reason is that the denial-of-service attack on the initial point prevents clients from receiving a proxy server IP address, effectively losing contact with the desired application servers.

1.2. Existing Approaches. Previously, the bootstrapping problem was tackled using two methods. One approach is (for example, MOTAG [2]) using proof-of-work [3–6] puzzles to force all prospective clients to solve computationally intensive puzzles before connecting to service in the initial point. This method utilizes the advantage of time, giving all prospective clients an equal chance to be served by the initial point. The disadvantage of proof-of-work puzzles is that a motivated attacker can establish a distributed denial-of-service attacks by controlling large botnets [7]. Also, proof-of-work puzzles slow down all clients, including the benign ones. The alternative approach (for example, DoSE [8]) is to mitigate the heavy traffic of prospective clients by installing the initial point in a content distribution network (CDN) and expanding the capacity of the network's initial point. Individual nodes in a CDN also require clients to solve puzzles to handle potentially overwhelming requests by clients.

1.3. Approach and Results. Our approach is to extend client bootstrapping by utilizing high-capacity networks such as notification networks provided by prominent cloud computing providers. We develop a protocol that uses a notification network as an initial point and performs full client bootstrapping that can handle a large number of client bootstrapping requests. The presented protocol maintains the secrecy of proxy servers, avoids unnecessary details in client bootstrapping, and is highly scalable. The implementation of the protocol is challenging and should satisfy the security and functionality requirements to be effective. Using prominent cloud-based solutions, we developed a prototype that can fully implement the protocol. Our prototype is developed for web applications with mobile clients.

In summary, the contributions of this work are

(1) A novel client bootstrapping model that uses high-capacity networks (such as notification services) as the facilitator for moving target defense systems in the cloud. The model enables a reliable client bootstrapping that improves the performance and the security of initial client interaction with the target network.

(2) A protocol for client bootstrapping in moving target defense systems for cloud-based networks.

(3) A full implementation of the system using Amazon Web Services and Google Cloud Platform with the prototype available as an open source project.

(4) A thorough security analysis of the presented model and a discussion of its applicability using currently available technologies.

(5) Evaluating the execution time for client registration using the presented bootstrapping model.

2. Related Work

Moving target defense systems rely on randomizing access to attack targets as the core technique to combat intrusions. Although all such systems share the common goal of distracting attackers from targets, the context in which moving target defense is applied can be distinct. For example, moving target defenses are designed to secure operating systems. One prominent example is the use of address randomization in the stack (e.g., [9–11]). Another area in which moving target defense is applied is distracting attackers from targets in a network. There are several previous works that have developed systems for securing classical networks (e.g., [2, 12–14]), software-defined networks (e.g., [15, 16]), and virtual cloud-based networks. Many of these proposals have common mechanisms, for example, randomizing the IP addresses, client authentication and registration, and techniques to shuffle clients. Techniques such as code watermarking [17] (inspired by [18]) can also be used for building deceptive systems that distract attackers from the target.

2.1. Moving Target Defense Solutions. Several classical works proposed creating relay networks that used client filtration to distract denial-of-service attackers from reaching their targets. These works were designed to work with physical networks without the use of software-defined networking or cloud computing platforms. Works such as Mayday [19], SOS [13], and WebSOS [12, 20] demonstrated the feasibility of the idea. Migrating OVErlay (MOVE) [21] was a subsequent work that reutilized network overlays with a rethinking of the client filtration. MOVE developed the idea of client-to-machine reassignments by introducing a technique for migrating application processes that run on victim machines to machines in regions of the network that were not affected by an attack. The new addresses were secret, and only authenticated clients were rerouted to the new machines. Badishi et al. [22] also proposed an attractive approach for classical networks in which moving target defense is realized through port hopping. The core idea is randomly selecting ports and redirecting clients to new ports. We share a similar technique in our model and distribute clients to random ports.

Similar to network overlays and client migrations, other works proposed the use of proxy servers as a primary mechanism for moving target defenses. The core idea is to limit access to application servers from intermediate proxy servers that authenticate and register clients. Then, filter client requests, only allowing requests from authenticated clients to reach application servers. MOTAG [2] (and similar works such as [23]) proposed assigning clients to single proxy servers, which is replaced by a pool of available proxy servers whenever the assigned proxy is found to be exposed. MOTAG proposed to shue clients when proxy servers were under attack (reactive shuing). Later, PROTAG [14] improved MOTAG's shuing policy by periodically reassigning clients to new machines regardless of their attack status, yet maintaining the mediation of secret proxy servers. Although PROTAG presents an interesting shuing policy, it does not address the problem of explicit attacks on authentication servers. The debate on optimal shuing policies and methods is continued in [14, 16, 24–26]. A recent study demonstrated ways to reduce attack detection time by utilizing resources on a victim machine [27]. However, our focus in this work is not on improving the shuing but provide an improved authentication method.

MTD in the cloud is achieved by utilizing the elasticity of cloud computing platforms. In a cloud environment, a customized virtual machine instance can be created in seconds by issuing API calls. Firewall rules and virtual routing, which are controlled and deployed in a cloud console, are also accessible from secure SDKs. The agility and elasticity of cloud tools have contributed to forming moving target defenses, as proposed by Jia et al. [28]. In [28], the network's surface is accessible to anonymous Internet clients. The core idea is to create replica servers on the fly, split the current traffic, and distribute it to the newly created servers. Our model avoided creating replicas as the primary moving mechanism and proposed to filter clients efficiently using the notification framework. Limiting access to ports at the level of the cloud computing platform eliminates unwanted traffic from anonymous clients. Brzeczko et al. approached the problem differently and proposed to analyze network traffic data in decoy cloud-based virtual machine instances and implemented a traffic redirection strategy based on the learned behavior [29]. Our model does not depend on data analysis as we do not explicitly filter clients. However, we can benefit from data analysis to introduce enhanced client filtration.

2.2. Client Bootstrapping. Although our focus is not on improving proof-of-work schemes, we discuss the details behind the idea by surveying some of the previous works. Aura et al. developed the basic security requirements for client puzzles [3]. Client puzzles should be easy to solve by the server, the solution to puzzles can be adjusted indefinitely, solutions cannot be searched in a database of precomputed solution candidates, and the same puzzle given to two different clients must require two distinct solutions. The authors suggest requiring the client to provide the value of x given $h(x)$, where $h(x)$ is a secure hash function. The work in [4] continues the effort and proposes a protocol that determines the server load and adjusts the puzzles accordingly. Waters et al. later argued that the puzzle scheme itself is the target of attacks [6]. Instead, they propose to link puzzles to server identity using public-key cryptography. This eliminates the possibility of forge puzzles. They also propose to have forward security, which prevents the use of puzzles used at a time the distributor of the puzzles was compromised. The work in [5] proposed the use of seeds in the puzzle generation algorithm and the distribution of seeds through DNS servers. The similarity of the work in [5] with our proposed model is in the idea of redirecting requests to a larger network. In our model, we use cloud-based notification networks as a cushion for the target server to slow down the traffic.

Content distribution networks promise to use a network of low-cost content distribution networks (CDN) as proxies to application servers, relying on the effectiveness of these networks [30]. The model reduces large-scale attacks on target applications [8, 31]. Although CDN proxies are a step forward for deploying moving target defenses for small and medium enterprises, they can potentially be a subject of residual resolution attacks. In these attacks, attackers can switch to unprotected platforms and reveal secret IP addresses behind CDN proxies [32]. While oblivious CDN [33] does not particularly address the problem of facilitating a moving target defense, it does complement CDN-based moving target defense systems. Oblivious CD provides a mechanism for preventing an attacker from learning about the content that it is delivering or the clients that it is serving. A potential alternative is to use a network of low-cost devices to develop a guarding scheme against initial attacks [34].

2.3. Open Problem. A summary of the related work is presented in Table 1. It remains an open problem to investigate a secure protocol for client bootstrapping using high-capacity networks. Although the use of content delivery network is a step towards solving the problem, a secure, light-weight, and simple to implement solution is highly desirable.

Table 1: A summary of related works.

Approach	Problem	Target cloud	Limitations
Relay networks	Client filtration	No	Requires thorough client identification
Cloud MTD	Client registration and shuffling	Yes	Requires thorough client identification or proof-of-work
Proof-of-work schemes	Client bootstrapping	No	Requires solution of computationally intensive puzzles
Content distribution networks	Client bootstrapping	Yes	Requires solution of computationally intensive puzzles or a large network

3. System Architecture

In this section, we start by presenting a general overview of our design (Section 3.2), followed by the description of our method for client registration (Section 3.3), and client-to-proxy assignments (Section 3.4). Finally, the security of our model is analyzed in Section 3.6.

3.1. Threat Model. An attacker is a remote program that sends application-layer requests to machines in a target network's surface (those machines that are accessible from the Internet). Benign clients send a reasonable number of requests in a time interval. A benign client does not represent a computational burden on the server. However, a client that sends an unusually high number of requests is also considered an attacker. The attackers aim to consume maximum resources from vulnerable machines in the target network. Attackers do not control the target network's surface and cannot escalate their privileges.

An unregistered (anonymous) client is the one that is not registered to connect to a machine in the target network. Except for those with black-listed addresses or identifiers, our model assumes all connecting clients are initially benign. However, the interaction is not directly with the moving target defense system. Thus, we assume none of the components of the moving target defense system are accessible from the Internet, except for proxy servers that respond to Internet requests.

3.2. Design Overview. The main goal of our model is to facilitate secure and DoS resilient client bootstrapping for a moving target defense system. In this article, we also refer to client bootstrapping as the client-to-proxy assignment problem. An unregistered client intends to communicate with an application service provided by the target network. The client communicates the intent to access the application server by registering with a notification service. The notification service is assumed to be a high-capacity network that can handle a large number of requests and has mechanisms to thwart denial-of-service attacks at various layers. The key idea is that the notification network mitigates the initial attacks, significantly reducing the burden on the target network. Conceptually, mitigating initial attacks using a high-capacity network is similar to the use of client puzzles as proposed in previous works. However, the advantage is that the target network does not have to accept initial connections at all. Thus, attack techniques by attempting to connect to the target network are completely mitigated.

When the client sends an initial request to the notification service, the registration process starts. The notification service registers the client and generates a unique notification token. The client automatically receives the notification token from the notification service (Figure 1). Once a notification token is generated for the client, the client's token and other identifiers are shared with the *controller*. The controller is a software component that runs in the target network and is not directly accessible from the Internet. The component continues communication with the client through the cloud notification service until the client is successfully registered or is denied access (Section 3.3).

The controller manages network access rules through the cloud computing provider. Each client is assigned to a specific proxy server through a port listener that is launched on the proxy server. The system grants communication access to the designated port only to clients assigned to a specific proxy. Proxy servers relay client requests and responses to and from application servers. A pool of proxy servers awaits client assignments. The machines in the proxy pool are used when the current *active* set of proxy servers reaches maximum capacity.

3.3. Registering Clients. An unregistered client is denied access to the target network's proxy servers. This is because proxy servers' IP addresses are only known to registered clients. Also, network access control only allows incoming traffic to a proxy from registered clients. We modify network access control policies using APIs provided by the cloud computing platform (for example, by modifying security groups in AWS).

3.3.1. Assignment Records. The controller (Figure 2) distinguishes between a registered client and an unregistered client, using *client-to-proxy assignment* records. If a valid record for a client exists, the client is registered, and the network access control policy is updated to allow the client to access the designated proxy server. A client-to-proxy assignment record (simply an assignment) $\mathscr{A}_i$ for a client i is a tuple:

$$\mathscr{A}_i = (T, C, A, P, L, t), \tag{1}$$

where T is a notification token received from the cloud notification service, C is a unique identifier (such as the International Mobile Equipment Identity that can be

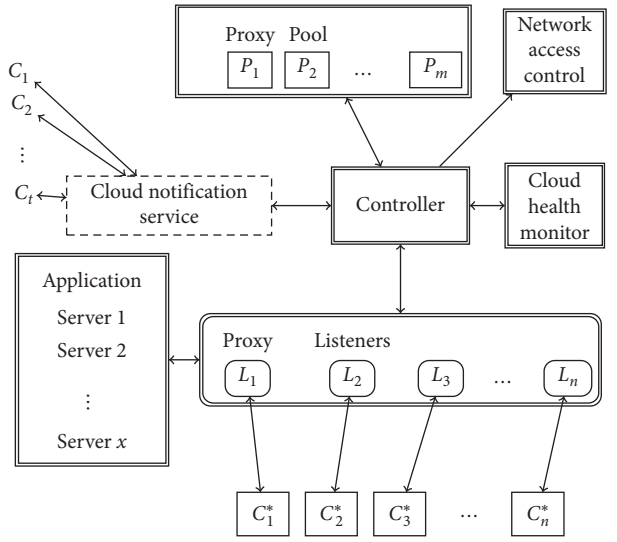

Figure 1: Architecture of a moving target defense system with a cloud-computing platform as a mediator. $C_1, \ldots, C_t$ are awaiting registration by communicating their intents through the notification service. The controller sends a client-proxy-assignment to each approved awaiting client using a notification. Proxy listeners only respond to registered clients $C_1^*, \ldots, C_n^*$. Application servers are not publicly accessible. A proxy machine in the proxy pool awaits assignments of clients when the number of clients surpasses client proxy capacities. Network access control rules are set by the controller to allow/deny access to proxies.

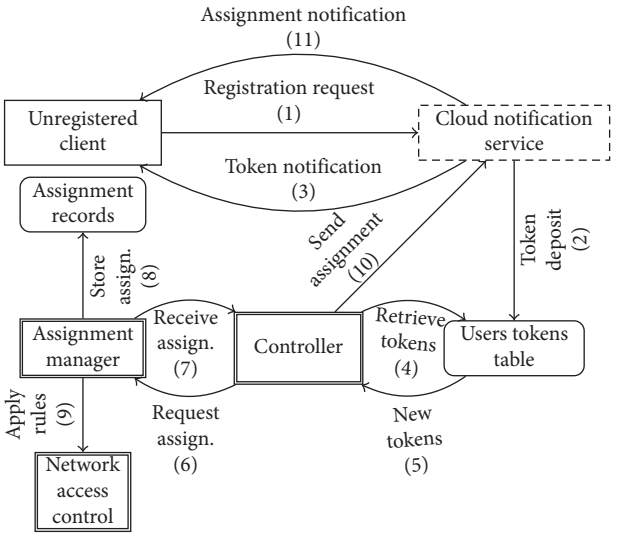

Figure 2: Processing a client's registration request through a notification network.

validated using the Lun algorithm [35, 36]) for the client's physical device, A is the client's IP address, P is the proxy's IP address, L is the listener port which limits inbound traffic to C, and t is a timestamp. Client-to-proxy assignment records (simply assignment records) are securely maintained in an internal database of the target network. The internal database is only accessible via the controller (which itself does not interact with Internet clients).

The notification token T is critical for the initial indirect communication between the client and the controller. Using the token T, a client can receive the client-to-proxy assignments, which must remain confidential. In typical usage of notification systems, clients register with the notification network, receive a token T, and present T to the application server to receive messages. This process is surprisingly insecure. A malicious client can intercept T and receive the victim's messages. To prevent this attack, the controller must *independently* receive T from a trusted cloud computing platform. Thus, in our model, the trusted cloud notification service distributes T to both the client and the controller.

3.3.2. Registration Process. As depicted in Figure 2, an unregistered client's registration process starts with registering with the notification service. The notification service is used as a mediator between the target network and the client. The notification service is independently managed by a cloud computing platform and is capable of handling and preventing denial-of-service attacks. Once a token T is generated, in Steps 2 and 3, T is stored in a database table shared with the controller and is transmitted to the requesting client. Both the cloud computing platform and the controller can read from and write to the table. The controller periodically checks for new client tokens in the shared table, as shown in Steps 4 and 5. Once a new client token is retrieved, the controller verifies if the client is not blocked from accessing the network. If the client passes the check, the controller requests from the *assignment manager* (Section 3.4) to create and store an assignment record for the client (Steps 6–9). Finally, in Step 10, the controller generates a notification message containing the record, which is sent through the cloud notification service in Step 11. The client receives the notification using a secure channel (such as Transport Layer Security).

3.4. Managing Assignments. This section explains the details of creating assignments, as shown in Steps 6–9 of Figure 2. A client assignment record $\mathscr{A}_i$ indicates successful registration of a client indexed i. The assignment manager is logically separated from the controller but can run as a stand-alone within the controller. The assignment manager creates and stores assignments, manages a pool of available proxies, and modifies network access control rules. At a high level, given client information, C and A, and the notification token T, the steps for creating a new assignment are given next. Let $\mathscr{Q}$ denote a queue of proxy servers that are ready to serve new clients. Then, to create an assignment,

(1) Dequeue a proxy P from $\mathscr{Q}$

(2) Dequeue a listener port L from $\mathscr{P}$

(3) Create an assignment record $\mathscr{A}$ using P, L, T, C, and A

(4) Send a message to the controller containing $\mathscr{A}$

(5) Modify the network access control rule to allow inbound traffic from client IP address A to proxy IP address P on port L

3.4.1. Assignment Policy. Our assignment policy addresses two concerns: the lifetime of assignments and proxy selection. Assignments are created and stored for unregistered clients and are never removed from assignment records. An assignment is modified when the client is *moved* to a new proxy. A single device C can be associated with multiple assignments, one for each IP A using which the client requests a new registration. However, among multiple assignments for the same client identifier C, only the assignment with the most recent timestamp is valid. With multiple assignments, a client can volunteer for self-reassignment, requesting a new registration and assignment using a new IP address, thus moving to a new proxy.

3.4.2. Proxy Addressing. For assignments to be used by Internet-based clients, a single public IP address must identify a proxy. The public IP address of a proxy P is only disclosed to those clients with an assignment to P. The IP addresses of the proxies are allocated individually by sending requests to the cloud computing platform's IP address allocation algorithm Ω. We assume that Ω behaves randomly and the IP addresses allocated by it are not predictable. If, in a specific cloud computing platform, addresses are predictable, an IP address shuffling scheme is required to increase IP address entropy. A predictable IP address allocation scheme defeats the primary purpose of moving target defense.

To optimize cost, our model requires two sets of proxies: an *active* proxy set π and an *awaiting* proxy set $\pi^\star$ (the proxy pool). When the system is bootstrapped, at least one active proxy must exist, that is, $|\pi| \geq 1$. The proxy pool, $\pi^\star$, is gradually expanded to include more proxies, based on the application demand. The specific algorithms for an optimized proxy pool $\pi^\star$ (for example, [37]) is out of the scope of this work.

3.4.3. Choosing a Proxy and a Listener. Clients are assigned to a listener L on a proxy P. Here, we discuss the problem of choosing a proxy and a listener for a client as a scheduling problem. The core requirement of a schedule for client assignments is to avoid choosing highly predictable proxies and listeners that. We consider predictability as to the probability of choosing a specific proxy for each new client assignment. A highly predictable choice is made when the probability of choosing a proxy is close to 1 and $|\pi|$ is minimized.

When $|\pi| = 1$, the choice of proxy is deterministic, to prevent the attacker from predicting the next client assignment, we increase the number of virtual machines available for assignments. With a single proxy, our model can reduce the probability of predicting the next assignment by randomizing the choice of the listener port on the proxy. We lower the predictability by assigning clients to specific ports. The predictability of assignments depends on the probability of choosing a proxy P and a listener C.

When $|\pi| > 1$, an unpredictable schedule for choosing a proxy is required. Our approach is to use a precomputed schedule for several future assignments. A schedule of proxy IP addresses is stored in a queue $\mathcal{Q}$. Initially, with an empty $\mathcal{Q}$, an IP address for each proxy in π is allocated by the cloud computing platform's Ω function and is stored in $\mathcal{Q}$. We assume that Ω does not produce predictable IP addresses. That said, IP addresses in $\mathcal{Q}$ are randomly shuffled. Other random shuffles of all values in $\mathcal{Q}$ are inserted in $\mathcal{Q}$, thus creating longer schedules. The assignment manager selects the IP addresses in $\mathcal{Q}$ as the choices of proxies for future assignments. Next, we continue to choose a listener port on the chosen proxy.

Similar to choosing a proxy, a listener port is chosen. Here, we create a proxy queue $\mathcal{P}$ filled with all randomized permitted port values (for example, all port values in range $[1023, 65535]$ for Linux). Each port is assigned to a single client. When the system is overloaded, serving clients more than available ports on all proxies, the schedule in $\mathcal{P}$ repeats and ports are assigned to more than one client.

3.4.4. Creating Assignments. For each client i, an IP address is dequeued from $\mathcal{Q}$ and inserted in a new assignment record $\mathcal{A}_i$. Similarly, a listener port L is dequeued from $\mathcal{P}$ and is inserted in $\mathcal{A}_i$. The assignment record is completed with the client identifier C and IP address A, the client's notification token (from the table shared with cloud notification service), and the current timestamp t.

3.4.5. Network Access Control. Modifying network access control rules are required to discard unwanted traffic to proxies. For each assignment, the assignment manager should update network access control rules to only allow inbound traffic from a specific list of client IP addresses. We either update software firewall rules (such as IPTABLES) on the virtual machine to control inbound traffic or update the network access control rules controlling network traffic through the cloud computing platform. The disadvantage of the first choice is that the attacker can attempt to establish connections, which is refused by the virtual machine. However, the attempted connections do consume resources, and the virtual machine itself is at the front line.

The second choice requires support from the cloud computing platform, which is possible in AWS using security rules or firewall rules in the Google Cloud Platform (GCP). Initially, all inbound traffic is denied using a general security rule. When an assignment record for a newly registered client is created, the assignment manager issues an API call to add a new security rule. The new security rule allows the client IP to connect on the chosen port L. The advantage of this approach is that the malicious traffic does not reach the targeted virtual machine if the origin's IP address is not white-listed.

Creating a new assignment relies on the implementation of the underlying data structures. To analyze the time complexity of the procedure, notice that Steps 1–4 require constant time. Step 5 involves a lookup in the access control list, which is handled by the cloud computing provider. In general, Step 5 requires a direct lookup, for example, using a dynamic array or a hash table, also requiring a single operation. Note that the time required to modify the access control list depends on network communication between the controller and the cloud

computing platform, and the performance of the data structure implemented in the cloud computing platform. In our experiments, as shown later in Section 5, the entire assignment procedure requires less than three seconds when 100 simultaneous clients attempt to register.

3.5. Complexity. In terms of time complexity, we analyze two major operations. One is client registration and the other is maintaining a list of banned clients that are detected to be attackers. Client registration requires maintaining a list of clients and their identifiers as generated by the controller. This list can grow very large as the number of clients grow. Adding a client to the ban list requires a single operation. However, as the client blacklist grows, registering new clients are affected. The client registration list and the blacklist are implemented as a binary tree, requiring a search of $O(\log n)$. Note that the tree is limited to the number of nodes it stores, preventing long search time. Thus, as new clients are added to the blacklist, the tree replaces existing nodes with new client information when the maximum number of nodes is achieved. Furthermore, the client lists can both be recorded on a relational database, taking the advantage of efficient column indexes for fast search.

3.6. Security Analysis. Our security model assumes a remote attacker that attempts to subvert the mechanisms provided in the design of the notification-based client bootstrapping. Here, we analyze the security of our model by considering

(1) Forging client identifiers
(2) Frequent registration requests
(3) Deterministic assignments
(4) Attacking without registration

3.6.1. Forging Client Identifiers. An attacker can attempt to forge client identifiers such as device identifier C, IP address A, or notification token T. By forging client identifiers, the attacker aims to deny access to a client by convincing the controller that the client is abusing the proxy. The attacker can also use forged identities to attempt large-scale distributed denial-of-service attacks. To achieve either of the two goals, the attacker must find the assigned proxy and present the required identities to bypass the controller's network access control rules. To forge a client identity, the attacker may attempt to obtain the notification token T. The attacker uses T for receiving the secret IP address of a proxy P to which the client is assigned. The attacker can intercept the communication between the client and the cloud notification service to intercept T. The effect of attack can be reduced by securing the communication channel using Transport Layer Security (TLS).

The attacker can also find P by eavesdropping on client communications. Then, the attacker extracts the IP address P (which is not encrypted in TLS) from network request headers, even when the communication is encrypted. For P to be useful, the attacker must also forge the client IP address A and the client's device identifier C. Forging A is feasible, but forging the specific device identifier C requires either a brute-force attack or compromising the client's device. Although there are theoretical possibilities, such attacks are not feasible, as they are complicated and require multiple successful attacks. Note that, such an attack can become more complicated by appending a unique cryptographically secure key K that is generated for the user and is presented with every network request. This extra K reduces the chance of a successful brute-force attack on an IMEI as the chosen value for C.

Since registration requires different pairs of IP addresses and device identifiers, an attacker can use a single IP address and forge multiple device identifiers. Obtaining actual device identifiers requires compromising legitimate client devices or using many devices, which are complicated and resource-intensive methods. An alternative is to generate device identifiers based on algorithms used to generate hardware identifiers for manufacturers. For example, an attacker can obtain the algorithm used to generate IMEI numbers and execute the algorithm to request multiple fake registrations. This attack can be thwarted in two days. First, the controller can impose a maximum number of IMEIs associated with a single IP address. Second, the cloud-based notification service should detect abusive and frequent requests to register for notification originating from an IP address. Even if the attacker attempts to use multiple spoofed IP addresses with fake IMEI values to register, the registration for the notification service fails. This is because the attacker cannot receive the notification tokens on a spoofed IP address.

3.6.2. Frequent Registration Requests. Our model does not limit the number of times a client can request a new registration. New registrations are limited to the exact combination of client IP address and device identifier. A client with access to dynamic IP addresses can request new registrations with a single device. Recall that the registration starts with requesting a notification token T. Requesting a new T slows down brute force attacks on the registration process because of the time required to execute the notification protocol. Even though registering with the notification service limits brute-force attackers, regular clients only require seconds (depending on the notification service provider) to register for notifications. After registering for notifications, the client cannot directly communicate with the controller. Thus, malicious clients cannot exhaust the controller's resources, for instance, by using frequent requests to register. Instead, when ready, the controller retrieves T based on a periodic schedule and proceeds to complete the client registration and produce an assignment.

3.6.3. Deterministic Assignments. A deterministic assignment is one that uses a predictable probability distribution to assign a client to a proxy such that an attacker can predict the next chosen proxy with considerable probability. The probability of correctly predicting the next assignment is dependent on the number of available proxy servers and the assignment scheduling policy. Consider m proxies that actively serve clients. With a round-robin schedule, starting

with proxy P_i, the attacker can be confident that the next proxy assignment to be P_{i+1}, knowing the current assignment P_i used by the system. In a target network with many client assignments, the attacker can be sure that the next assignment is on P_{i+j} for some $j > i$. With a uniform random schedule, the chance of selecting any proxy is $1/m$. Thus, increasing the number of proxies is crucial to lowering the probability of correctly predicting assignments.

Predicting assignments helps the attacker to concentrate efforts on specific proxy servers. As mentioned earlier, we do not focus on improving the scheduling policy (as discussed in [38, 39]). This is a significant problem that requires further investigation in future work.

3.6.4. Attacking without Registration. Attackers may attempt to deny service to benign clients without registering in the system. We assume attacking the notification service is not useful to the attacker as the cloud computing provider is powerful and capable of thwarting the attack. To attack the proxy servers directly, the attacker should identify the IP addresses of the proxies. Identifying the proxies could use the probabilistic analysis mentioned earlier. Assuming the attacker can identify IP addresses of the proxies, the attacker must target the cloud computing provider. This is because our model utilizes the network access control rules of the cloud computing provider. Every open port has a specific rule that only allows inbound traffic from clients assigned to the port. The attacker can spoof the client IP addresses to bypass the rule. Such an attack requires precise prediction of one or a few IP addresses that are allowed to use the port. This attack is a time consuming and resources intensive effort that does not seem to be practical.

4. System Implementation

We developed a prototype system representing the notification-based model introduced in Section 3.2. Our prototype consists of an Android client, several Firebase tables, and an implementation of the controller. The controller is implemented using software development kits (SDKs) from Firebase and AWS, primarily in Python and Node.js. The primary functions of the prototype include all the required services from the controller as described earlier in Section 3. The source code is available on https://github.com/kussl/nMTD.

Given current technologies, the choice of Firebase for sending notifications is critical to our model (as described later in this section). We hosted the proxies and the controller server on AWS. However, the required functionality of the proxies and the controller can also be implemented entirely in Firebase. Our goal is to demonstrate the feasibility of our model using currently available technologies. The implementation of our model faces challenges in

(1) The choice of cloud notification service and implementation of the architecture and security requirements of new client registrations

(2) Assigning a client to a random port number in the selected proxy

(3) Dumping network requests to a cloud-based file system for future investigation of malicious requests

4.1. Implementing New Client Registrations. We studied two primary cloud notification services: Apple Push Notification service (APNs) and Firebase Cloud Messaging. Both services have similar registration requirements for clients. After successful registration, the client and the application server are given a client notification token T. When the application server wants to send a notification to the client, it passes T and a message M to the notification service. Locally, the application server associates T with a client identifier to be used for client-specific messages. The critical difference between the two services is in the way the application server receives the notification token T.

Unfortunately, APNs do not provide a way for the application server to receive the notification token directly from APNs servers. Instead, the protocol requires the client to give T to the application server. This violates our model, which does not trust the client with T since a malicious client can present a stolen T. We chose Firebase Cloud Messaging, which provides a secure way to share T with the application server. The registration process in Figure 2 is precisely implemented in Firebase. Once a client is registered with Firebase Cloud Messaging, T is generated and is recorded on a table. The application server can reside either on Google Cloud Platform or any other cloud-based hosting service (thus, connecting to the table using an API call). The table of tokens cannot be accessed by any client or any other component in the target network. Inbound traffic is only allowed from the IP address of the controller server.

4.2. Assigning a New Client to a Proxy. Assigning a new client to proxy is in six steps: (1) retrieve a new token, (2) create a new client record, (3) choose a proxy server and port number, (4) launch a listener on the selected proxy server and port number, (5) generate a message for the client, and (6) send a notification to the client containing the generated message.

First, the controller runs an event listener in the background to detect a new client notification token, which triggers client registration. The new client token is recorded in the Tokens Table. The controller verifies if the client is blocked from registration, by searching the client identifiers in a blocklist table. Note that our prototype does not include methods for the detection of suspicious clients at the time of registration. The controller registers for a Node.js listener that is triggered by Firebase when a new client record appears in the Tokens Table. The registered listener in the controller receives a new_token record, which could be processed immediately. With several new proxy servers added, one is chosen by random, followed by a random choice of the port number. The unused port numbers are stored in a buffer from which the new port number is selected. With a selected port on a proxy, the controller moves to launch a new listener port. When the port is successfully launched, the security setting on the selected proxy is

updated. The rules could be updated using the AWS command line tools:

aws ec2 authorize-security-group-ingress --group-id GROUP_ID --protocol tcp --port SELECTED --cidr CLIENT,

where GROUP_ID is the selected proxy's group identifier (used for a group of EC2 instances in AWS), SELECTED is the selected port by random, and CLIENT is the new client IP address being registered. Note that, in AWS, security rules can be applied to groups of virtual machines. Thus, in our implementation, each group only has a single proxy, since each proxy has different security rules.

Finally, a new notification is created, containing the selected proxy's IP address and the launched listener port, which is sent to the newly registered client.

4.3. Dumping Network Requests. Analyzing network requests to an overloaded proxy helps in detecting malicious clients. The challenge is that overloaded proxy servers cannot respond to request when under attack. Thus, there are two ways to monitor proxies. First, to know if a proxy is under heavy load, we use the AWS CloudWatch service to detect high CPU or memory usage in all proxy machines. We developed a daemon that checks CPU and memory usage for each proxy machine every minute. As of this writing, the necessary information from CloudWatch is provided free of charge. Detailed monitoring of usage patterns is available with additional charges.

Our model requires analyzing the access pattern on specific ports. Thus, we needed to develop a specific monitoring system to detect which ports are receiving the highest network requests. The hypothesis is that ports with more requests demand higher CPU and memory (regardless of the type of request used). This hypothesis can be reevaluated in other implementation to include detailed measurements. Our approach is to periodically dump network requests to an external file system that can be retrieved and analyzed by the controller. This functionality can either be implemented by dumping network requests to an AWS Simple Cloud Storage Service (S3) Bucket using FUSE https://github.com/s3fs-fuse/s3fs-fuse.git, which mounts an AWS S3 Bucket to a Linux machine. Alternatively, one can mount a remote directory on an independent Linux machine that is internally connected to all proxy servers. We chose to develop a daemon that runs on each proxy, reads Apache forensic log files http://httpd.apache.org/docs/2.2/mod/mod_log_forensic.html (containing network requests for the application), and transfers them to an AWS S3 Bucket using AWS APIs.

5. Execution Time

In this section, we present the performance of our approach by examining the time required to register and assign a new client to a proxy and the time required to reassign an existing client to a new proxy. Our model assumes that clients connect using mobile devices. Thus, for our experiments, we developed a web-based mobile client simulator that can register for cloud notifications and receive a new assignment through a notification. Note that experimentation with large mobile clients is not feasible as the protocol requires full registration of clients using valid IMEI numbers. Hence, the numbers below are for the clients that could be simulated using Google Firebase tools on a single machine.

5.1. Client Registration

5.1.1. Setup. The Android client initiates a request to register with the controller after receiving registering with the notification server. Then, the controller creates a database record for the client, after randomly selecting a proxy IP address and port. In this experiment, all the steps enumerated in Section 4 are executed, starting with sending a request to receive a notification and ending with receiving a notification containing a proxy assignment. The time measured includes the effort to generate a notification token by Firebase Cloud Messaging service. The registration time involves choosing a random port number, launching a new listener port, and requesting AWS to modify security rules to restrict inbound traffic to the requesting client on the chosen port.

5.1.2. Results. As Figure 3(a) depicts the results for registering and assigning a new client to a proxy. The experiment tests register several simultaneous clients. The trend line shows a polynomial trend in increasing the time required for registration. As the figure shows, when the number of clients is increased by a factor of 10, the time required is only increased by 54%. In contrast, solving computationally intensive puzzles are intended to delay clients up to several tens of seconds. The reason is that puzzles are intended to slow down the requests. For example, the puzzle proposed in [4] requires computing x, given the preimage obtained from $h(x)$, where $h(x)$ is a preimage resistant function. This operation is supposed to delay the client by T time units and suffers from two weaknesses. First, all clients, including the benign nodes, are delayed. Second, a motivated attacker can reduce the search time by separating the computation. Our scheme has a modest performance penalty, enforces proper registration without sacrificing privacy, and efficiently delays flooded messages without affecting benign clients.

5.2. Client Reassignment

5.2.1. Setup. In this experiment, the performance penalty of reassignment is measured. As described earlier in Section 4, the reassignment is triggered when a high CPU usage is detected. Once the controller is notified of this event, we start measuring the time until the client receives a new IP address and port combination. We assume that the client manually changes its connection when it receives a new IP address and port combination. Thus, we disconnect the client after a timeout, which is also sent to the client.

5.2.2. Results. The results of Figure 3(b) depicts the effort required for the process of reassigning *all* clients to new proxy servers. The reassignment involves closing the current

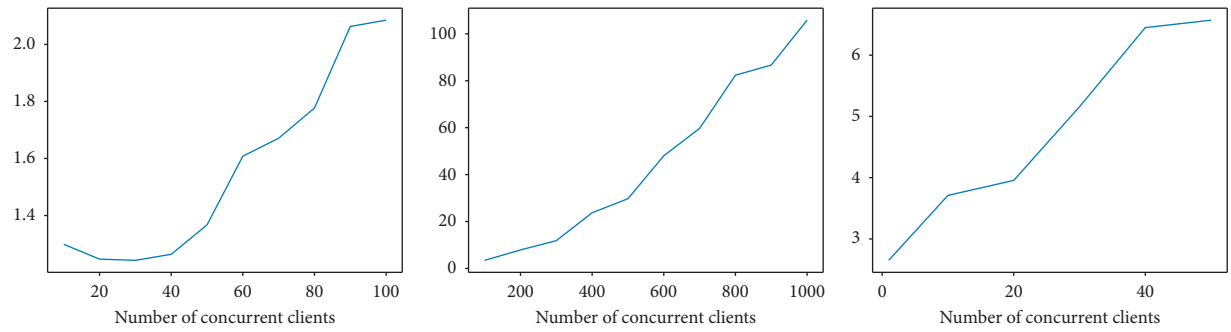

Figure 3: Execution time in seconds for (a) registering new concurrent clients, (b) reassigning clients to new proxies, and (c) reassigning clients to new proxies under a synthetic DoS attack. The number of clients are actual clients with IMEI numbers.

listener port, launching a new listener port, and modifying the security rules in AWS. The client is informed about the new assignment with a notification message. Measuring reassignment at a large scale, Figure 3(b) shows the total time required to reassign several hundreds of simultaneous clients. As the figure shows, reassigning 1000 simultaneous clients approximately requires 105 seconds of time.

5.3. DoS Attack

5.3.1. Setup. In this experiment, the effect of a denial-of-service attack is tested. The assumption is that there are currently 100 active clients registered with the controller and connected to their respective proxies. In this case, the experiment tests the time performance of detecting, reassigning, and moving the clients to new proxies. In this scenario, the attacking clients are detected based on analyzing the log of requests on each port. If a port has caused too many requests or has exploited a vulnerability, it is marked as an attacker assignment (precise detection is outside the scope of this work and has been explored heavily in previous works). The attacking ports remain on the same IP addresses, quarantined and deceived, and all other benign clients are reassigned to new IP addresses.

5.3.2. Results. We developed a synthetic vulnerability that could be exploited by attackers. When exploited, a vulnerable URL causes unusual system activity on the target. This mimics the numerous software vulnerabilities that are exploited daily. Figure 3(c) shows the performance of reassigning clients in our implementation when attackers are present. The time shown is the total duration for all 100-x clients to receive new IP addresses, where $x \in [1, 50]$ is the number of active trackers.

5.4. Comparison with Existing Solutions.

Here, we compare the execution time of our work with four existing work chosen to reflect various alternative models to the one presented in this work. The results of the comparisons are in Table 2. The table shows the time per client as presented with the type of operations measured. Because of the differences in the focus, various closely related operations are considered. The performance of our model (client bootstrapping) is compared in two settings: new client registration (Figure 3(a)) and client reassignment (Figure 3(b)). The closest performance is by Portcullis [5], which uses various schemes including client puzzles. The worst performance is with CDN redirection in [30] and CDN-on-demand [31]. Catch-me-if-you-can does not provide an alternative to client registration. However, it presents a closely related approach with the performance time measured in client migration. As shown in the results, client bootstrapping provides a competitive advantage in terms of execution time, which includes the entire process of the client bootstrapping protocol.

Table 2: Comparison of the presented client bootstrapping time performance against alternative solutions.

Model	Time per client (s)	Operation
Client bootstrapping	0.026	New client registration
Client bootstrapping	0.034	Client reassignment
Portcullis	0.040	Capability setup
Catch-me-if-you-can	0.083	Client migration time
CDN-on-demand	1.172	Serving clients
CDN redirection	2.370	Response latency

6. Conclusion

This paper presents a client bootstrapping protocol for moving target defense, which reduces the time required to register new clients in a cloud-based network. The presented protocol aims to mitigate denial-of-service attacks on initial network entry points. Existing methods mainly use client puzzles for mitigating attacks on the initial network entry points. These methods require several seconds of *intentional delay* for clients to solve computationally intensive puzzles before allowing the client to connect. This slow down is to prevent attacks on the first encounter with the clients. However, our solution replaces unnecessary delays with the mandatory notification registration, which could be implemented in less than three seconds as the results suggest. Thus, the achievement maintains the security promises of moving target defenses while preventing delays for benign clients. Our model is limited to clients who can acquire

unique identifiers such as the IMEI number. This limitation can be potentially addressed using client fingerprinting techniques, which is the subject of future work. Furthermore, intrusion detection systems (such as [40]) can also assist moving target defense system to lower the volume of initial attacks.

Data Availability

Our results are available on https://github.com/kussl/nMTD.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] G. F. Lyon, "Nmap network scanning: the official nmap project guide to network discovery and security scanning," 2009.

[2] Q. Jia, K. Sun, and A. Stavrou, "MOTAG: moving target defense against internet denial of service attacks," in *Proceedings of the 2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9, Nassau, Bahamas, July 2013.

[3] T. Aura, P. Nikander, and J. Leiwo, "Dos-resistant authentication with client puzzles," in *Security Protocols*, B. Christianson, J. A. Malcolm, B. Crispo, and M. Roe, Eds., Springer, Berlin, Germany, pp. 170–177, 2001.

[4] D. Dean and A. Stubblefield, "Using client puzzles to protect tls," *USENIX Security Symposium*, vol. 42, 2001.

[5] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu, "Portcullis: protecting connection setup from denial-of-capability attacks," *ACM SIGCOMM Computer Communication Review*, vol. 37, pp. 289–300, 2007.

[6] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten, "New client puzzle outsourcing techniques for dos resistance," in *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pp. 246–256, Washington, DC, USA, October 2004.

[7] F. C. Freiling, T. Holz, and G. Wicherski, "Botnet tracking: exploring a root-cause methodology to prevent distributed denial-of-service attacks," in *European Symposium on Research in Computer Security*, pp. 319–335, Springer, Berlin, Germany, 2005.

[8] P. Wood, C. Gutierrez, and S. Bagchi, "Denial of service elusion (DoSE): keeping clients connected for less," in *Proceedings of the 2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*, pp. 94–103, Montreal, Canada, September 2015.

[9] S. Bhatkar, D. C. DuVarney, and R. Sekar, "Address obfuscation: an efficient approach to combat a board range of memory error exploits," in *USENIX Security Symposium*, Washington, DC, USA, August 2003.

[10] L. Li, J. E. Just, and R. Sekar, "Address-space randomization for windows systems," in *22nd Annual Computer Security Applications Conference*, Miami Beach, FL, USA, December 2006.

[11] P. Team, "PaX address space layout randomization (ASLR)," 2003, https://pax.grsecurity.net/docs/aslr.txt.

[12] D. L. Cook, W. G. Morein, A. D. Keromytis, V. Misra, and D. Rubenstein, "Websos: protecting web servers from ddos attacks," in *The 11th IEEE International Conference on Networks, 2003 ICON2003*, pp. 461–466, Sydney, Australia, September 2003.

[13] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: an architecture for mitigating DDoS attacks," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 176–188, 2004.

[14] 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 *2016 IEEE Conference on Communications and Network Security (CNS)*, pp. 198–206, Philadelphia, PA, USA, October 2016.

[15] A. Chowdhary, S. Pisharody, and D. Huang, "SDN based scalable MTD solution in cloud network," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense, MTD'16*, pp. 27–36, New York, NY, USA, 2016.

[16] Y.-H. Lin, J.-J. Kuo, D.-N. Yang, and W.-T. Chen, "A cost-effective shuffling-based defense against HTTP DDoS attacks with SDN/NFV," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, Paris, France, May 2017.

[17] C. Iwendi, Z. Jalil, A. R. Javed et al., "KeySplitWatermark: zero watermarking algorithm for software protection against cyber-attacks," *IEEE Access*, vol. 8, pp. 72650–72660, 2020.

[18] M. T. Ahvanooey, Q. Li, X. Zhu, M. Alazab, and J. Zhang, "Anitw: a novel intelligent text watermarking technique for forensic identification of spurious information on social media," *Computers & Security*, vol. 90, Article ID 101702, 2020.

[19] D. G. Andersen, "Mayday: distributed filtering for internet services," in *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4, USITS'03*, Berkeley, CA, USA, 2003.

[20] W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Misra, and D. Rubenstein, "Using graphic turing tests to counter automated ddos attacks against web servers," in *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS'03*, pp. 8–19, New York, NY, USA, 2003.

[21] A. Stavrou, A. D. Keromytis, J. Nieh, V. Misra, and D. Rubenstein, "MOVE: an end-to-end solution to network denial of service," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2005*, San Diego, CA, USA, 2005.

[22] G. Badishi, A. Herzberg, and I. Keidar, "Keeping denial-of-service attackers in the dark," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 3, pp. 191–204, 2007.

[23] H. Wang, Q. Jia, D. Fleck, W. Powell, F. Li, and A. Stavrou, "A moving target ddos defense mechanism," *Computer Communications*, vol. 46, pp. 10–21, 2014.

[24] C.-C. Liu, B.-S. Huang, C.-W. Tseng, Y.-T. Yang, and L.-D. Chou, "Sdn/nfv-based moving target ddos defense mechanism," in *Advances in Intelligent Systems and Computing*, pp. 548–556, Springer International Publishing, Cham, Switzerland, 2019.

[25] Y. Shan, G. Kesidis, and D. Fleck, "Cloud-side shuffling defenses against ddos attacks on proxied multiserver systems," in *Proceedings of the 2017 on Cloud Computing Security Workshop, CCSW'17*, pp. 1–10, New York, NY, USA, 2017.

[26] Y. Yang, V. Misra, and D. Rubenstein, "A modeling approach to classifying malicious cloud users via shuffling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 2, pp. 6–8, 2019.

[27] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and M. Rajarajan, "Scale inside-out: rapid mitigation of cloud ddos attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 959–973, 2018.

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

[29] A. Brzeczko, A. S. Uluagac, R. Beyah, and J. Copeland, "Active deception model for securing cloud infrastructure," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 535–540, Toronto, Canada, April 2014.

[30] L. Wang, V. Pai, and L. Peterson, "The effectiveness of request redirection on cdn robustness," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 345–360, 2002.

[31] Y. Gilad, A. Herzberg, M. Sudkovitch, and M. Goberman, "Cdn-on-demand: an affordable ddos defense via untrusted clouds," in *23rd Annual Network and Distributed System Security Symposium, NDSS 2016*, San Diego, CA, USA, February 2016.

[32] L. Jin, S. Hao, H. Wang, and C. Cotton, "Your remnant tells secret: residual resolution in DDoS protection services," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 362–373, Luxembourg City, Luxembourg, June 2018.

[33] A. Edmundson, P. Schmitt, N. Feamster, and J. Rexford, "OCDN: oblivious content distribution networks," 2017, http://arxiv.org/abs/1711.01478.

[34] M. Numan, F. Subhan, W. Z. Khan et al., "A systematic review on clone node detection in static wireless sensor networks," *IEEE Access*, vol. 8, pp. 65450–65461, 2020.

[35] B. Causley, "The secret behind the Luhn-ie," *XRDS: Crossroads, the ACM Magazine for Students*, vol. 19, no. 1, pp. 81-82, 2012.

[36] H. Luhn, "Computer for verifying numbers," 1960.

[37] J. L. L. Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Dynamic placement of virtual machines for cost optimization in multi-cloud environments," in *2011 International Conference on High Performance Computing Simulation*, pp. 1–7, Istanbul, Turkey, July 2011.

[38] R. Colbaugh and K. Glass, "Predictability-oriented defense against adaptive adversaries," in *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2721–2727, Seoul, Republic of Korea, October 2012.

[39] M. L. Winterrose and K. M. Carter, "Strategic evolution of adversaries against temporal platform diversity active cyber defenses," in *Proceedings of the 2014 Symposium on Agent Directed Simulation, ADS'14*, pp. 9:1–9:9, San Diego, CA, USA, 2014, http://dl.acm.org/citation.cfm?id=2665049.2665058.

[40] S. Bhattacharya, S. R. K. S, P. K. R. Maddikunta et al., "A novel pca-firefly based xgboost classification model for intrusion detection in networks using gpu," *Electronics*, vol. 9, no. 2, pp. 219–235, 2020.