MOTAG: Moving Target Defense Against Internet Denial of Service Attacks

Quan Jia
George Mason University
Fairfax, Virginia 22030
qjia@gmu.edu

Kun Sun
George Mason University
Fairfax, Virginia 22030
ksun3@gmu.edu

Angelos Stavrou
George Mason University
Fairfax, Virginia 22030
astavrou@gmu.edu

***Abstract*—Distributed Denial of Service (DDoS) attacks still pose a significant threat to critical infrastructure and Internet services alike. In this paper, we propose *MOTAG*, a moving target defense mechanism that secures service access for authenticated clients against flooding DDoS attacks. *MOTAG* employs a group of dynamic packet indirection proxies to relay data traffic between legitimate clients and the protected servers. Our design can effectively inhibit external attackers' attempts to directly bombard the network infrastructure. As a result, attackers will have to collude with malicious insiders in locating secret proxies and then initiating attacks. However, *MOTAG* can isolate insider attacks from innocent clients by continuously "moving" secret proxies to new network locations while shuffling client-to-proxy assignments. We develop a greedy shuffling algorithm to minimize the number of proxy re-allocations (shuffles) while maximizing attack isolation. Simulations are used to investigate *MOTAG*'s effectiveness on protecting services of different scales against intensified DDoS attacks.**

***Index Terms*—DDoS; Moving Target Defense; Secret Proxy; Insider; Shuffling**

I. Introduction

Arbor Networks has reported a significant increase in the prevalence of large-scale distributed denial-of-service (DDoS) attacks in recent years [1]. In 2010, the largest reported bandwidth achieved by a flood-based DDoS attack reached 100 Gbps. Meanwhile, the cost of performing a DDoS attack has turned out to be surprisingly low. A Trend Micro's white paper [2] has revealed that the price for 1-week DDoS service could be as low as $150 on Russian underground market.

A number of mechanisms have been proposed in the past to prevent or mitigate DDoS attacks. Filtering-based approaches [3], [4], [5] use ubiquitously deployed filters to block unwanted traffic sent to the protected nodes. Capability-based defense mechanisms [6], [7], [8], [9] endeavor to constrain the resource usage by the senders within the threshold permitted by the receivers. Secure overlay solutions [10], [11], [12], [13], [14], [15] interpose an overlay network to indirect packets between clients and the protected nodes, aiming to absorb and filter out attack traffic. However, these static defense systems either rely on global deployment of additional functionalities on Internet routers or require large, robust virtualized network to withstand the ever-exacerbating attacks. Besides, some of them are still vulnerable to sophisticated attacks, such as sweeping [11] and adaptive flooding attacks [12].

In this paper, we propose *MOTAG*, a dynamic DDoS defense mechanism that adopts moving target defense strategy to protect centralized online services. In particular, *MOTAG* offers DDoS resilience for authorized and authenticated clients of security sensitive services such as online banking and e-finance. *MOTAG* employs a layer of secret moving proxies to mediate all communications between clients and the protected application servers. The network-level filters surrounding the application servers only allow traffic from the valid proxy nodes to reach the protected servers.

Proxy nodes in *MOTAG* have two important characteristics. First, all proxy nodes are "secret" in that their IP addresses are concealed from the general public and are exclusively known by legitimate clients after successful authentication. Each legitimate client is provided with the IP address of one working proxy at any given time to avoid unnecessary information leakage. We apply existing proof-of-work (PoW) schemes [16], [17], [18], [19] to protect the client authentication channel. Second, proxy nodes are "moving". As soon as an active proxy node is attacked, it is replaced by another node at a different location, and the associated clients are migrated to alternative proxies. We show that these characteristics not only enable us to mitigate brute-force DDoS attacks, but also empower us to discover and isolate malicious insiders that divulge the location of secret proxies to external attackers. We do so via shuffling (repositioning) clients' assignment to new proxy nodes when their original proxies are under attack. We develop algorithms to accurately estimate the number of insiders and adjust client-to-proxy assignment accordingly to rescue most innocent clients after each shuffle.

Our solution does not rely on global adoption on Internet routers or collaboration across different ISPs to function. Neither do we depend on resource-abundant overlay network to out-muscle high bandwidth attacks and to provide fault tolerance. Instead, we take advantage of our proxies' secrecy and mobility properties to fend off powerful attackers. This entails lower deployment costs while offering substantial defensive agility, resulting in an effective DDoS protection.

II. Threat Model and Assumptions

Instead of targeting open and general-purpose web services, we focus on protecting security sensitive online services against *network flooding attacks*. The clients of the protected services are pre-authorized and their identities can be authenticated before they are served. We assume a large pool of backup proxies that attackers are incapable of attacking altogether. However, only a small group of proxies are active at any time to avoid extensive operational costs. An ideal source for the proxy pool is one or several cloud environment where customers are charged only for running instances.

We assume powerful attackers with high aggregate bandwidth that are capable of simultaneously overwhelming many standalone machines on the Internet. However, we do not assume attackers that can saturate well-provisioned Internet backbone links for ISPs, data centers, and cloud service providers. Attackers, in case of uncertainty, can first perform a reconnaissance attack (e.g., IP and port scanning) to pinpoint targets for the subsequent flooding attack.

With the knowledge of *MOTAG* mechanism, attackers can also flood the authentication channel through which the legitimate clients are admitted. However, it is significantly harder for attackers to pass strong authentication by brute force and reach the proxies as legitimate clients. To uncover the network location of proxies, some attackers may plant "insiders" by compromising legitimate clients or eavesdropping on legitimate clients' network connections. However, the number of such insiders in a protected system will be limited.

III. MOTAG Architecture

Figure 1 shows the overall architecture of *MOTAG*, which consists of four inter-connected components: the authentication server, the proxies, the filter ring, and the application server. The application server provides the online services (e.g., banking or e-finance services) that we want to protect and make accessible to authenticated clients. The IP address of the application server is concealed from all clients. The proxy nodes are a group of dynamic and distributed machines that relay communications between clients and the application server. The filter ring, similar to what was described in [12], is comprised of a number of high speed routers placed around the application server, allowing inbound traffic only from valid proxy nodes. The authentication server is responsible for authenticating clients and assigning legitimate ones to individual proxy nodes.

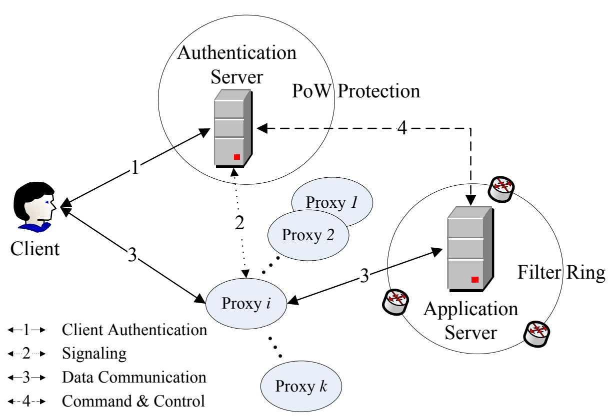


Fig. 1. Overview of the MOTAG Architecture.

MOTAG allows a client to access the application server only if the client can be successfully authenticated. One simple solution is to associate the application domain name with the IP address of the authentication server during DNS registration. Each successfully authenticated client will be randomly assigned to one of the active proxy nodes whose identities are not publicly known. The authentication server will inform each client about the IP address of a designated proxy, and in the meantime, notify the proxy node about the forthcoming connection from the client. The authentication server, as well as each proxy node, maintains a dedicated interface for the purpose of signaling. Through this signaling channel, proxies report to the authentication server if they are attacked; the authentication server informs proxies about client assignment and coordinate their actions against DDoS attacks.

The authentication server also assigns a *capability token* for each client-to-proxy session. This token limits a client's throughput by specifying the number of packets (or, the number of bytes) allowed for the session in the next time window (t seconds). A proxy should receive identical copies of a capability token from two parties for every session, one from the authentication server to notify new client assignment and one from the client as a proof of identity. Every proxy node maintains a per-session counter and regulates traffic according to individual capability. Such capability-based policing is key for detecting external, brute-force flooding attacks in that it distinguishes authorized packets from illegal ones. Furthermore, it can detect and frustrate any internal attempt to abuse the assigned capability such as sharing capability with external attackers. For communications between proxy nodes and the application server, a lightweight authenticator as described in Mayday [12] can be employed for proxy identity validation. The filter ring routers can perform fast lookups to verify such lightweight authenticators in proxy-to-application packets. These authenticators can be dynamically altered and active proxy nodes will receive timely updates via the signaling channel. To prevent the authentication server from being flooded by botnets, we employ proof-of-work (PoW) schemes [16], [17], [18], [19] to ensure its accessibility for legitimate clients.

A. Secret Moving Proxies

In *MOTAG*, we build an intermediate layer with a pool of geographically distributed proxies to diffuse attackers' traffic and shield the application server behind. We do not activate all proxies at once. Instead, we only keep a small subset of proxies working at all times and dynamically substitute attacked ones at runtime, confusing attackers with "moving" proxies. The IP addresses of all proxies are concealed from the general public. Our proxies are resilient to scanning attacks because they only talk to IP addresses of the authenticated clients. The mapping from clients to working proxies is many-to-one.

If a proxy node is under attack, it will be shut down and a new proxy node at a different network location will be activated for replacement. Proxy substitution is a fast, lightweight operation because all proxies run the same, simple traffic indirection logic and maintain no client state. All session information is centrally stored at the application server. A few hot proxy spares can be kept alive over time and quickly kick in whenever necessary. All clients connecting to the attacked node will be re-assigned across the entire set of active proxies. The new assignment can be pushed to the affected clients by the authentication server, or the clients can be re-authenticated for security assurance. We name the overall process of proxy replacement and client re-allocation as client-to-proxy *shuffling* and will introduce the details in Section IV. No shuffling will be performed if there is no attack, and a small set of proxy nodes with constant IP addresses are used to serve all legitimate clients.

MOTAG is different from existing overlay network solutions [10], [11], [12], [15], which rely on a fairly static network composition of overlay nodes to tolerate and filter out the attack traffic. Building and maintaining such an overlay entails extensive and continuous investment to acquire more nodes and bandwidth. In addition, sweeping [11] and adaptive [12] flooding attacks may cause severe service disruptions. In contrast, *MOTAG* keeps its proxies confidential and mobile. Only authenticated clients are informed about their assigned proxies. This enhances defense agility against massive, sophisticated attacks while reducing its dependence on the volume of proxy resources.

B. Authentication with Proof-of-Work Protection

The authentication server with assured accessibility is essential to our moving target defense. It acts as the initial checkpoint to separate legitimate clients from illegal ones. We use authentication as a mechanism to bind a client to a specific network flow. Only with such unique binding, we are able to keep track of the behavior of each client throughout the shuffling process. Every client has to pass authentication before assigned to a working proxy that eventually routes traffic to the application server. The IP addresses of authenticated clients are recorded and sent to proxies to enforce IP-based filtering. The authentication server is also responsible for advertising subsequent client-to-proxy assignments during shuffling. *MOTAG* is agnostic to the specific authentication mechanism employed.

The authentication server is the only part of the *MOTAG* architecture that can be publicly addressed. Therefore, it can be a new target of distributed flooding attacks. To mitigate this attack, we take advantage of existing proof-of-work (PoW) schemes [16], [17], [18], [19], which force clients to solve cryptographic puzzles before allowing them to consume resources on the server side. In particular, they can realize per-computation fairness regarding bandwidth usage among all clients [19], prevent connection depletion attacks [18], and mitigate DDoS attacks on application-level authentication protocols [16], [17]. Although mandating extra computational task can help reduce attackers' throughput, it also imposes considerable burden on legitimate clients as well. Therefore, PoW approaches are suitable for protecting client authentication in that authentication packets are infrequently sent and are more delay-tolerant. However, they are less preferable for securing application data communication due to its high overhead.

IV. Client-to-Proxy Shuffling

Hiding proxies while enforcing client authentication can effectively prevent external attackers from reaching *MOTAG*'s packet delivery system. Moreover, by keeping proxies mobile and performing guided shuffling on client-to-proxy assignments, *MOTAG* can also mitigate insider attacks that expose secret proxies to flooding attacks.

Attackers can implant malicious insiders in the targeted system via social engineering, compromising legitimate clients, stealing clients' identities for authentication, and eavesdropping on clients' network connections. Installed insiders in a protected system are the results of targeted attacks with relatively high technical sophistication. Thus, the number of functioning insiders is expected to be small (maybe hundreds). Nevertheless, the damage they can cause is still significant. Once insiders uncover the IP addresses of some proxy nodes, they will notify external attackers who will carry out DDoS attacks against these exposed proxies. We address such attacks as insider-assisted DDoS attacks, or simply insider attacks. Although insider attacks cannot be fully prevented, we aim to minimize their impact on innocent clients. In this paper, we design a client-to-proxy shuffling mechanism to quarantine insider attacks over time and ensure service accessibility for as many innocent clients as possible.

A. Shuffling Strategy

In *MOTAG*, a pool of proxy nodes are reserved and idled before DDoS attacks break out. As soon as an attack happens, a small number of proxy nodes in the pool are activated. The set of active proxy nodes can be logically classified into two groups, namely *serving proxies* and *shuffling proxies*. Serving proxies provide more reliable connection services to the known innocent clients, while shuffling proxies are responsible for shuffling operations and only provide intermittent connections to suspicious clients. When attacked, shuffling proxies will be replaced and the associated clients are flushed and reassigned.

At the beginning, all the active proxies are unmarked. All clients are randomly assigned to proxies. Each client will be assigned to only one proxy at a time. If some proxies are attacked after the initial assignment, they will be marked as shuffling proxies while others are considered serving proxies. By employing the greedy algorithm described in Section IV-C, we repeatedly shuffle the client-to-proxy assignment within the shuffling proxy group to distinguish insiders from innocent clients and segregate them.

After each shuffle, some shuffling proxies will still be attacked and some will not. The intact shuffling proxies become serving proxies and the associated clients are marked as trusted and considered as saved from the on-going attack. Clients connected to the attacked proxies are considered untrusted, since we cannot tell who are the actual insiders within this group. To save the innocent ones among them, we will randomly re-distribute all the untrusted clients across the group of shuffling proxies. Given the specific number of suspicious clients and available proxy nodes, new proxies can be activated as shuffling proxies from the pool to help accelerate shuffling operations. Generally speaking, the more shuffling proxies are available, the faster insiders will be quarantined.

By repeating the client-to-proxy shuffling for multiple rounds and keeping record of the suspicious proxies/clients, we can narrow down the range of suspects and gradually identify most innocent clients. The insiders will eventually be quarantined and the attack damage will be minimized.

Notice that the shuffling process is stateless, meaning each shuffle is considered independent. The tags (trusted/untrusted) we place on clients will be reset after each shuffle, to avoid being confused by insiders' inconsistent behavior. These tags do not necessarily reflect the true identity of the clients. Plus, the roles of proxies (shuffling/serving) are interchangeable across shuffles, depending on the behavior of attackers. The goal of shuffling operations is to separate innocent but attacked/suspected clients from true insiders. Although some insiders may make us believe in their innocence by staying inactive, we can ensure that they are not going to cause extra damage if they begin to attack later.

B. Shuffling Optimization

To contain insider attacks as quickly as possible, and also to adapt to system dynamics such as client mobility, we need a shuffling algorithm that can identify and separate as many innocent clients as possible per shuffle. To that end, we first analyze the number of innocent clients to save under different client-to-proxy assignments.

Specifically, among a total number of N clients to be shuffled, the number of insiders is N_i, and the number of innocent clients is N_c, so we have $N_i + N_c = N$. [1] After one round of shuffling, N_{ca} innocent clients are still being attacked, and N_{cu} of them are not ($N_{ca} + N_{cu} = N_c$). Our goal is to mathematically compute the expected value of N_{cu} (denoted as $E(N_{cu})$) under different circumstances and find a way to maximize it, given a number of K available shuffling proxies. We use A_j to represent the number of clients appointed to proxy j.

Obviously, $E(N_{cu}) = \sum_{j=1}^{K} p_j A_j$, where p_j is the probability that proxy j is not being attacked. Considering an arbitrary proxy j, it is not being attacked only when none of the insiders are connecting to it. Hence, p_j is also the probability that all insiders are assigned to proxy nodes other than j. According to simple combinatorics, $p_j = \binom{N-A_j}{N_i} / \binom{N}{N_i}$, where $\binom{N}{N_i}$ is the total number of ways to distribute the N_i insiders within the population N, and $\binom{N-A_j}{N_i}$ is the number of combinations that all insiders are within the $N - A_j$ clients not connecting to proxy j. Therefore, the expected value of N_{cu} can be calculated by Equation IV.1.

$$E(N_{cu}) = \sum_{j=1}^{K} p_j A_j = \frac{\sum_{j=1}^{K} \binom{N-A_j}{N_i} A_j}{\binom{N}{N_i}} \tag{IV.1}$$

We also have $E(N_{ca}) = N_c - E(N_{cu})$.

Given the total number of clients N, the number of insiders N_i, the number of shuffling proxies K, and the client-to-proxy assignment vector $\mathbf{A}$, we want to maximize $E(N_{cu})$. Intuitively, the more shuffling proxies are used, the more innocent clients are expected to be saved via each shuffle. In the extreme case where $K \geq N$, each client can be allocated with an exclusive proxy node ($A_j = 1, \ \forall j \in (1, K)$). $E(N_{cu}) = N_c$ means no innocent client will be attacked. This is the ideal scenario where all insiders are quarantined their own proxy nodes within one round of shuffling. However, in practice, it is usually impossible to provide a dedicated proxy node for each client when clients are large in number. In most cases, the client population would outnumber the shuffling proxies by far ($K << N$). Consequently, the way of distributing clients across proxy nodes becomes utterly important.

Assuming we have a constant number of K shuffling proxies, we are facing an optimization/maximization problem with Equation IV.1 being the objective function. The variables are summarized into the vector $\mathbf{A}$ of natural numbers that defines the client-to-proxy assignment scheme, with the constraint being

$$\sum_{j=1}^{K} A_j = N, \ \ where \ \mathbf{A} \in \mathbb{N}^K \tag{IV.2}$$

Although recursive algorithms such as dynamic programming can be employed to compute the optimal solution, we adopt a greedy approach here to produce a quick and near-optimal solution. Our simulations under various configurations show that the results produced by the greedy algorithm approach very closely to the theoretical upper bound of $E(N_{cu})$.

C. The Greedy Shuffling Algorithm

Algorithm 1 shows the greedy algorithm for computing the client-to-proxy assignment. The main function is called *GreedyAssign*. Since in Equation IV.1 $E(N_{cu})$ is the sum of pieces (i.e. $p_j A_j$) for all shuffling proxies computed in the same way, we firstly perform optimality analysis for an individual component. For an arbitrary proxy j, A_j can be any value within $[0, N-1]$. A_j cannot be N. Otherwise, everyone will be attacked if there is an insider onboard.

Since the value of N_i will affect the optimal choice of A_j, for a particular N_i, we enumerate all possible values of A_j and select the one (ω) that maximizes $p_j A_j$. This subroutine is described in procedure *MaxProxy* of Algorithm 1. Under our greedy approach, we assign ω clients to as many proxies as possible.

Function *GreedyAssign* is called recursively to assign the remaining clients to the rest of the proxies. The computation will terminate under three conditions. First, when there are more proxy nodes left than clients, each client will be assigned to an exclusive proxy node. Second, when there is only one proxy left, all remaining clients will be appointed to it. Third, when the expected number of remaining insiders is rounded to 0, all remaining clients will be evenly distributed for load balancing. The overall computational complexity of the greedy algorithm is $O(N * N_i)$. To further reduce the computational overhead throughout the shuffling procedures, the client-to-proxy assignment vectors for different N, K, N_i combinations

[1] We provide a method to estimate the number of insiders in Section IV-D.

can be pre-computed and stored in lookup tables for runtime reference.

Algorithm 1 Greedy algorithm for computing client-to-proxy assignment.

function GREEDYASSIGN($Client, Insider, Prox$)
  if $Client \leq Prox$ **then**
    Assign 1 exclusive proxy to each client
  else if $Prox = 1$ **then**
    Assign all clients to the proxy
  else if $Insider = 0$ **then**
    Evenly distribute $Client$ over $Prox$
  else
    $\omega = MaxProxy(Client, 0, Client-1, Insider)$
    $ProxToFill = floor(Client/\omega)$
    if $ProxToFill \geq Prox$ **then**
      $ProxToFill = Prox - 1$
    $RemC = Client - ProxToFill * \omega$
    $RemP = Prox - ProxToFill$
    $RemA = Round(\frac{Insider * RemC}{Client})$
    Fill $ProxToFill$ Proxies with ω clients each
    Fill the rest proxies according to
    $GreedyAssign(RemC, RemA, RemP)$

procedure MAXPROXY($Client, Lbnd, Ubnd, Insider$)
  Max=0, MaxAssign=0
  for $i = Lbnd \rightarrow Ubnd$ **do**
    $Save = \binom{Client-i}{Insider} i / \binom{Client}{Insider}$
    if $Save > Max$ **then**
      $Max = Save,\ MaxAssign = i$
  return MaxAssign

To evaluate the optimality of the greedy algorithm, we will compare its results with the theoretical upper bound of $E(N_{cu})$. Since Equation IV.1 is a summation of $p_j A_j$ for each individual shuffling proxy j, the max of IV.1 cannot be greater than the sum of the max of each $p_j A_j$ when relaxing Constraint IV.2, i.e. $Max(E(N_{cu})) \leq K * Max(p_j A_j)$. Here, $Max(p_j A_j)$ can be obtained by running subroutine $MaxProxy(N, 0, N-1, N_i)$. The comparison between the greedy algorithm and the theoretical upper bound is done via simulations under various configurations on MATLAB. The results are presented in Section VI.

D. Estimating the Number of Insiders

In our earlier discussion, we assume the number of insiders (N_i) is fixed and given; however, in practice, we have no such prior knowledge. Since the value of N_i has direct influence on the client-to-proxy assignment, it is important to make accurate estimation. In addition, such estimation has to be made at each shuffle to cope with varying insider number and their behavior.

We solve this problem using maximum-likelihood estimation (MLE). We first establish a connection between the number of insiders N_i and the number of proxies that are not under attack (denoted as X). In a particular attack where $X = m$, we calculate the probabilities $Pr(X = m)$ with regard to different N_i values, and use the N_i value that maximizes the probability as the estimated number of insiders.

According to the inclusion-exclusion principle under balls-and-urns model [20], we can compose Equation IV.3 to calculate $Pr(X = m)$, where $Pr(X \geq M)$ stands for the probability that at least M ($M = m, m+1, \ldots, K$) proxy nodes are not attacked, K is the total number of all shuffling proxies.

$$Pr(X=m) = Pr(X \geq m) - \binom{m+1}{m} Pr(X \geq (m+1)) + \binom{m+2}{m} Pr(X \geq (m+2)) - \ldots + (-1)^{K-m} \binom{K}{m} Pr(X \geq K) \quad \text{(IV.3)}$$

In particular, these M not-under-attack proxies constitute the set $\mathbf{U} = \{u_1, u_2, \ldots, u_M\}$, where u_j is the real ID of the jth available proxy node. Set $\mathbf{U}$ can be any M sized subset of the K shuffling proxies.

The key idea to compute $Pr(X \geq M)$ is similar to how we derive Equation IV.1. If a particular set $\mathbf{U}$ of proxies are not attacked, the insiders must be among the clients assigned to the rest proxy nodes (the complement of $\mathbf{U}$). Thus, we have Equation IV.4, in which $\sum_{\mathbf{U}}^{(M)}$ denotes the summation over all possible combinations of $\mathbf{U}$ (all M sized subsets of the K shuffling proxies), and $N - \sum_{j=1}^{M} A_{u_j}$ gives the number of clients connecting to the proxies not in $\mathbf{U}$. u_j is an arbitrary proxy node in the set, and A_{u_j} denotes the number of clients assigned to that node.

$$Pr(X \geq M) = \frac{\sum_{\mathbf{U}}^{(M)} \binom{N - \sum_{j=1}^{M} A_{u_j}}{N_i}}{\binom{N}{N_i}} \quad \text{(IV.4)}$$

Under a certain client-to-proxy assignment scheme $\mathbf{A}$, we can now correlate $Pr(X = m)$ with N_i by combining Equation IV.3 and IV.4.

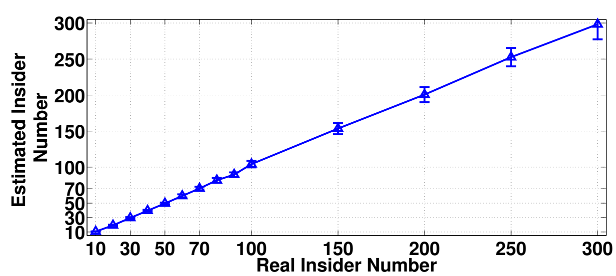


Fig. 2. Insider estimation under 10K clients, 100 shuffling proxies

To evaluate our insider estimation algorithm, we implement it on MATLAB and run simulations with different numbers of insiders. Based on the number of attacked proxies, our algorithm makes educated guesses on the real insider numbers. The guess that maximizes the result of Equation IV.3 becomes the final estimation. These estimations are plotted against true insider numbers, in Figure 2. For each data point, we run the simulation 30 times to compute the mean and 99% confidence interval. According to the results in the figure, our algorithm gives very accurate estimations.

V. Security Analysis

MOTAG is a dynamic traffic indirection framework opened only to authenticated clients. It is designed to account for both brute force and sophisticated DDoS attacks against the protected application server and other potential targets.

a) **Resistance to brute-force attacks:** With *MOTAG* protection, external attackers will not be able to locate secret proxies without planting insiders, i.e. compromising or eavesdropping on legitimate clients. Even if attackers somehow get to know the IP range of the entire proxy pool, they are still unable to find out which proxies are currently active. *MOTAG* is also invulnerable to scanning attacks thanks to IP-based filtering. All active proxies will only respond to IP addresses representing legitimate clients who have been successfully authenticated. By outsourcing proxy infrastructure to one or more cloud providers, the proxy pool (which can be the entire cloud domain) will be so large that even powerful botnets are unable to attack altogether. In the meantime, we only pay for the running proxy instances over time.

The mobility of proxy nodes adds another layer of resiliency against brute-force attackers. In the case that attackers hit a secret proxy by chance, the attacked node will quickly "move away". Without the ability to trace the shifting proxies, external attackers will get lost in front of the moving targets.

The only exposed component of the system is the authentication server. It is protected by existing PoW schemes that ensures legitimate clients can eventually get authenticated given reasonable efforts and delay.

b) **Resistance to insider attacks:** Malicious insiders pose a more serious threat because they have access to secret moving proxies. Basically, Insiders can expose the proxies that they uncover to the horizon of a powerful bonet. As discussed in Section IV, by dynamically "moving" proxy nodes and shuffling client-to-proxy designation optimally, *MOTAG* can have such insiders quarantined in a few rounds. The precious bandwidth of the application server wasted on attackers are at most proportional to the ratio of proxies under attack (rather than the number of attackers), which will drop throughout multiple rounds of shuffles. In addition, since the shuffling decisions are specifically made for each round, *MOTAG* can easily accommodate to system dynamics such as the arriving and leaving of clients and insiders. Some insiders may stay silent across shuffles, aiming to profile the IPs of the proxy pool. However, since proxies that are not under attack will remain stable and the associated clients will not be shuffled, silent insiders will stay on the same proxy forever and fail their purpose.

c) **Resistance to compromised proxies:** With the help of malicious insiders, attackers may even compromise some proxy nodes. If successful, the application server and the authentication server will be directly exposed to attackers. However, lightweight authenticators, similar to what was used in [12], can be used to identify and filter proxy-to-server traffic. Therefore, the compromised proxies that are exploited to attack the application or authentication server can be readily identified. Their packets will be blocked by the high-speed filter ring deployed around the servers.

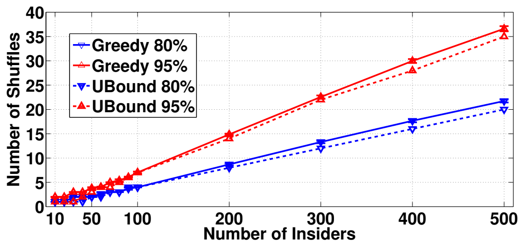
VI. MOTAG Evaluation

A. Insider Quarantine Capability

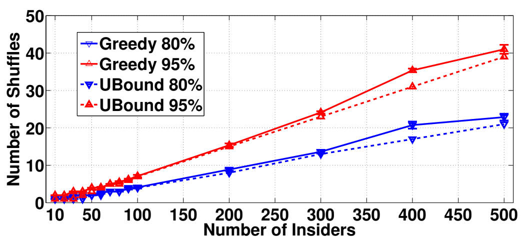
We experimentally evaluate *MOTAG*'s effectiveness on mitigating insider-assisted DDoS attacks. To that end, we implement all core algorithms of *MOTAG* on MATLAB and run them with simulated clients and proxy nodes. Although *MOTAG* allows the change on client and insider population between different shuffles, we keep the numbers constant for the easy of experimentation. In our simulations, we randomly pick clients to be malicious insiders without informing *MOTAG*. *MOTAG* decides the number of clients assigned to each proxy node but each client is randomly appointed to a proxy with empty slots. We use Mersenne twister [21] as our random number generator. We assume insiders will always attack. We also assume attackers possess infinite bandwidth. Therefore, all proxies connected by insiders are under attack. However, as discussed earlier, we only assume a limited number of insiders (hundreds) considering the difficulty to bypass strong authentication. *MOTAG* uses the MLE method in Section IV-D to estimate the number of existing insiders and uses the greedy algorithm in Section IV-C to determine the client-to-proxy assignment for the next shuffle. It usually takes more than one shuffle to save a majority of innocent clients when the number of insiders is large. Figure 3 quantitatively shows the number of shuffles needed to save 80% and 95% innocent clients by using our greedy algorithm (solid lines) and by applying the theoretical upper bound of Equation IV.1 (dotted lines) in each shuffle. Figure 3a and 3b vary the number of insiders while keeping the total number of clients and shuffling proxies constant. Figure 3c and 3d only change the number of shuffling proxies. $10,000$ clients are simulated in Figure 3a and 3c, while $100,000$ clients are simulated in Figure 3b and 3d. We run the same 30 times simulation for each data point and plot with 99% confidence interval.

First, we see that in almost all cases, the performance of *MOTAG* is close to the theoretical optimum. This means that greedy algorithm is very close to optimal. Then, Figure 3a and 3b show that the number of shuffles needed to save the same percentage of innocent clients grows almost linearly with the increase in the number of insiders. More shuffles indicate longer time to mitigate an attack, but it also means that attackers have to devote much more effort to recruit more insiders. Figure 3c and 3d reveal that the number of necessary shuffles increases as less proxy nodes are available. The lines climb slowly when the proxies outnumber the insiders and become significantly steeper otherwise. Moreover, the narrow confidence intervals of *MOTAG*'s data points indicate that the performance of our shuffling algorithm is reliable and predictable.

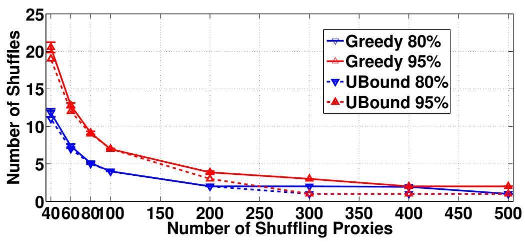
Notice that the change from $10,000$ to $100,000$ clients almost causes no difference in the simulation results. Instead, the ratio between the number of shuffling proxies and the number of insiders is the decisive factor on protecting innocent



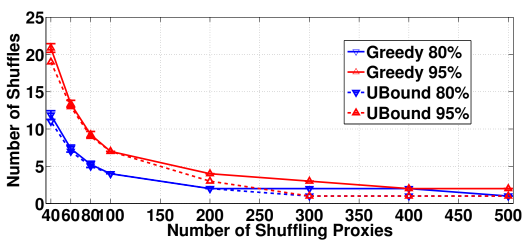
(a) Varying the number of insiders under 10K clients, 100 shuffling proxies



(b) Varying the number of insiders under 100K clients, 100 shuffling proxies



(c) Varying the number of shuffling proxies under 10K clients, 100 insiders



(d) Varying the number of shuffling proxies under 100K clients, 100 insiders

Fig. 3. The number of shuffles needed to save 80% and 95% of innocent clients

clients. Figure 4 shows the minimum number of shuffling proxies required to save 95% of innocent clients within 5, 10, and 15 shuffles, respectively. The number of insiders ranges from 10 to 800. The solid lines represent a client population of 10 thousand and the dotted lines denote 100 thousand. We see a close to linear relationship between the number of required shuffling proxies and the number of insiders in achieving a constant security goal. These results can help system administrators decide how many proxy nodes they will need to achieve their security goals. Again, a 10 fold increase in the client population only has a minor impact on the results and the 99% confidence intervals are almost negligible.

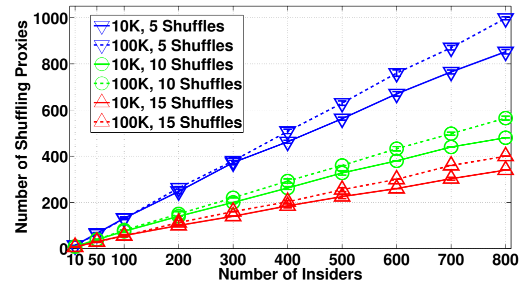


Fig. 4. Number of proxy nodes needed to save 95% of innocent clients within 5, 10, and 15 shuffles, with 10K and 100K clients and a increasing number of insiders

B. Overhead

MOTAG mainly introduces two aspects of overhead to communications between clients and the application server, namely proxy-based communication indirection, and client-to-proxy shuffling.

TABLE I
LATENCY OVERHEAD INTRODUCED BY PROXY INDIRECTION

	Direct	Indirect			
	RTT	Mean RTT	Overhead	Max RTT	Overhead
1	63ms	104ms	63.35%	143ms	125.41%
2	86ms	99ms	15.64%	128ms	49.45%
3	83ms	102ms	23.73%	133ms	60.47%
4	90ms	112ms	23.77%	131ms	45.18%
5	84ms	107ms	27.73%	120ms	42.48%

TABLE II
THROUGHPUT OVERHEAD INTRODUCED BY PROXY INDIRECTION (MB/S)

	1	2	3	4	5
Direct	90.66	83.46	86.24	123.30	121.20
Indirect	15.20	14.46	13.99	15.97	14.09

First, to assess the overhead introduced by proxy-based traffic indirection, we select 10 geographically distinct U.S. nodes from PlanetLab to form 5 end-to-end flows. We also randomly pick 24 other nodes that spread across the country to serve as proxies. We measure the latency and throughput for both direct and indirect communications of the 5 flows, and the results are shown in Table I and Table II, respectively. SSH tunneling through individual proxy node is employed to redirect traffic between end nodes. Round trip time (RTT) numbers are obtained by bouncing short TCP messages back and forth between the end nodes of each flow 100 times to get the mean. Throughput numbers are the average of 10 Iperf [22] sessions. Apparently, the impact of introducing proxies on latency (usually less than 30%) is much less significant than its influence on throughput. The drop on throughput is not only caused by traffic indirection by proxies, but is also a result of message encryption and decryption by SSH agents. In fact, different crypto strategies, including no encryption, can be listed as options when implementing *MOTAG* based systems.

Users can make informed decisions based on the nature of the protected application.

TABLE III
TIME TO SWITCH BETWEEN TWO PROXY NODES (SECONDS)

	1	2	3	4	5
MEAN	0.514	0.512	0.509	0.546	0.530
MAX	0.677	0.773	0.693	0.714	0.753
MIN	0.291	0.208	0.249	0.357	0.214

The time needed to shuffle clients among different proxy nodes determines the agility and usability of *MOTAG* against insider attacks. Quick shuffles will make it harder for attackers to "follow" and have insiders quarantined faster. At the same time, innocent but shuffled clients will suffer less severe service disruptions. Therefore, to quantify the impact of our system to the end users, we measure the time needed for a client to switch from one proxy node to another. To that end, we choose 5 geographically dispersed nodes from PlanetLab to be the destination servers. We randomly pick another node to play the role of the authentication server. We time the entire process that our local client gets notified by the authentication server, then discards the current proxy and connects to the new proxy, until reaching back to the destination server. During this process, the authentication server sends a session ticket to both the client and the new proxy node, the client will present this ticket to the proxy to get authenticated. Only after that, the new proxy node will start forwarding packets for the client. We use another 8 PlanetLab nodes as proxies and switch between them. The average, maximum, and minimum proxy switching times for each destination are listed in Table III. The numbers are fairly small yet consistent. Less than one second proxy switching time should not cause significant service disruption for most non-realtime applications.

VII. RELATED WORK

A number of research efforts have been devoted to defense against DDoS attacks over the past decade [23]. Filtering-based approaches [3], [4], [5] intend to use ubiquitously deployed filters to block unwanted traffic far away from the protected nodes. They assume that attack traffic can be differentiated from legitimate traffic. However, this is usually a difficult job because attackers can sneak through by spoofing IP addresses and mimicking normal senders. Instead of trying to distinguish and then block malicious traffic, *MOTAG* first performs client authentication to filter out illegal clients. Only authenticated clients will be appointed to the secret moving Internet proxies that can directly talk to the protected application server.

Capability-based mechanisms adopt a different philosophy that gives the control over resource usage to the packet receiver [6], [7], [8], [9]. Senders have to obtain receivers' explicit permission before sending packets to them. Traffic from authorized or privileged senders with valid capability can be prioritized during an attack. Using capability is a more proactive way of defense. Nevertheless, such solutions also rely on a global adoption on the Internet routers for adequate capability enforcement, which is unlikely to happen given limited incentives. *MOTAG* uses capability token to identify and rate-limit authenticated clients. Rather than depending on high degree of deployment on the Internet routers, we employ a thin layer of secret moving proxies for traffic policing.

To eliminate the physical network constraints and administrative boundaries, secure overlay networks are proposed to provide flow authentication, filtering, indirection, as well as attack tracking and tolerance [10], [11], [12], [13], [14], [15] on top of the Internet. The common goal is to hide the protected nodes behind the well-provisioned, distributed overlay network that is capable of absorbing DDoS traffic. TOR [24] is a well-known implementation of overlay network. By using an exposed, relatively static overlay network to withstand the ever-intensifying DDoS attacks inflicted by expanding botnets, the defenders will involve themselves in a never-ending armed race with the attackers. Even if a strong overlay network that can tolerate any DDoS attacks is in place, advanced attackers can start by attacking a small portion of the overlay nodes and sweep through the entire overlay step by step [11]. By repeating such sweeping attack, attackers are guaranteed to hit the critical nodes and cause major service disruptions. Sophisticated attackers can even measure the impact of their attacks via recruited legitimate clients. They can use such feedback to spot and hence adapt their attack to focus on the pinch points [12]. Moreover, the protected server can potentially be exposed via insider attacks [25].

Besides overlay network, there are other efforts that hide the paths to selected services behind intermediate protections [26], [27]. These solutions intend to employ a simpler, easier-to-deploy protection layer to filter out un-authorized traffic and are thus conceptually similar to *MOTAG*. Unfortunately, they fail to account for attacks in which authorized clients act as malicious insiders to compromise their interlayer protection. In this paper, we thoroughly analyzed insider threats and proposed a shuffling mechanism to quarantine insider attacks.

MOTAG endows mobility to its packet indirection proxies. This resembles the earlier network address randomization technique against hitlist worms [28] and the fast-flux scheme to sustain accessibility to illegal commercial websites [29]. To the best of our knowledge, we are the first in using such dynamic method on defense against DDoS attacks.

VIII. ACKNOWLEDGEMENTS

This work is supported by the United States Defense Advanced Research Projects Agency (DARPA) through Contract FA8650-11-C-7190. Opinions, findings, conclusions and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US Government, or DARPA.

IX. CONCLUSION

We present *MOTAG*, a framework that employs dynamic, hidden proxies as moving targets to mitigate network flooding DDoS attacks. To reach the protected service, authenticated clients are assigned to individual proxy nodes that perform packet forwarding and session policing. When a DDoS attack is mounted against *MOTAG* proxies, the authenticated clients connected to the attacked proxies are re-assigned to alternative proxies at realtime, enabling them to evade the ongoing attack and maintain access the protected service. With *MOTAG*, we can effectively hide the protected critical services from external attackers. Sophisticated attackers can only use insiders to locate our proxy nodes and attack them. *MOTAG* employs a novel, efficient shuffling mechanism to quarantine insider-assisted attacks. Our simulations show that *MOTAG* can protect a majority of innocent clients from DDoS attacks assisted by hundreds of insiders within a small number of shuffles. In addition, our experimental methodology and the results can be used to guide the implementation and deployment of *MOTAG*-based DDoS defense systems.

References

[1] R. Dobbins and C. Morales, "Worldwide infrastructure security report vii," 2011. [Online]. Available: http://www.arbornetworks.com/report

[2] T. Micro, "Russian underground 101," http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-russian-underground-101.pdf, 2012.

[3] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *ACM Computer Communication Review*, vol. 32, pp. 62–73, 2002.

[4] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing," RFC 2827 (Best Current Practice), Internet Engineering Task Force, May 2000, updated by RFC 3704.

[5] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: network-layer dos defense against multimillion-node botnets," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. New York, NY, USA: ACM, 2008, pp. 195–206.

[6] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing internet denial-of-service with capabilities," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 39–44, 2004.

[7] A. Yaar, A. Perrig, and D. Song, "Siff: A stateless internet flow filter to mitigate ddos flooding attacks," in *IEEE Symposium on Security and Privacy*, 2004, pp. 130–143.

[8] X. Yang, D. Wetherall, and T. Anderson, "Tva: a dos-limiting network architecture," *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1267–1280, 2008.

[9] X. Liu, X. Yang, and Y. Xia, "Netfence: preventing internet denial of service from inside out," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 255–266. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851214

[10] A. D. Keromytis, V. Misra, and D. Rubenstein, "Sos: Secure overlay services," in *Proceedings of ACM SIGCOMM*, 2002, pp. 61–72.

[11] A. Stavrou and A. D. Keromytis, "Countering dos attacks with stateless multipath overlays," in *Proceedings of the 12th ACM conference on Computer and communications security*, ser. CCS '05. New York, NY, USA: ACM, 2005, pp. 249–259. [Online]. Available: http://doi.acm.org/10.1145/1102120.1102153

[12] D. G. Andersen, "Mayday: distributed filtering for internet services," in *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*. Berkeley, CA, USA: USENIX Association, 2003, pp. 3–3.

[13] R. Stone, "Centertrack: an ip overlay network for tracking dos floods," in *SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2000, pp. 15–15.

[14] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang, "dfence: Transparent network-based denial of service mitigation," in *NSDI*, 2007.

[15] C. Dixon, T. Anderson, and A. Krishnamurthy, "Phalanx: withstanding multimillion-node botnets," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 45–58. [Online]. Available: http://dl.acm.org/citation.cfm?id=1387589.1387593

[16] T. Aura, P. Nikander, and J. Leiwo, "Dos-resistant authentication with client puzzles," in *Security Protocols Workshop*, 2000, pp. 170–177.

[17] D. Dean and A. Stubblefield, "Using client puzzles to protect tls," in *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, ser. SSYM'01. Berkeley, CA, USA: USENIX Association, 2001, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251327.1251328

[18] 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*, ser. CCS '04. New York, NY, USA: ACM, 2004, pp. 246–256. [Online]. Available: http://doi.acm.org/10.1145/1030083.1030117

[19] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu, "Portcullis: Protecting connection setup from denial-of-capability attacks," in *Proceedings of the ACM SIGCOMM*, August 2007.

[20] N. Johnson and S. Kotz, *Urn Models and Their Applications: An Approach to Modern Discrete Probability Theory*. New York: Wiley, 1977, ch. 1.3.2.

[21] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998. [Online]. Available: http://doi.acm.org/10.1145/272991.272995

[22] "Iperf," http://iperf.sourceforge.net.

[23] M. Abliz, "Internet denial of service attacks and defense mechanisms," University of Pittsburgh, Tech. Rep. TR-11-178, Mar 2011.

[24] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *In Proceedings of the 13 th Usenix Security Symposium*, 2004.

[25] L. Overlier and P. Syverson, "Locating hidden servers," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, ser. SP '06, Washington, DC, USA, 2006, pp. 100–114.

[26] V. Kambhampati, C. Papadopoulos, and D. Massey, "Epiphany: A location hiding architecture for protecting critical services from ddos attacks," in *The 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, 2012.

[27] X. Wang and M. K. Reiter, "Wraps: Denial-of-service defense through web referrals," in *25th IEEE Symposium on Reliable Distributed Systems*. IEEE Computer Society, 2006, pp. 51–60.

[28] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis, "Defending against hitlist worms using network address space randomization," in *Proceedings of the 2005 ACM workshop on Rapid malcode*, ser. WORM '05. New York, NY, USA: ACM, 2005, pp. 30–40. [Online]. Available: http://doi.acm.org/10.1145/1103626.1103633

[29] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks," in *NDSS*. The Internet Society, 2008.