Proactive Server Roaming for Mitigating Denial-of-Service Attacks

Sherif M. Khattab*, Chatree Sangpachatanaruk§, Rami Melhem*, Daniel Mossé* and Taieb Znati*§
*Department of Computer Science
§Department of Information Science and Telecommunications
University of Pittsburgh, PA 15260
{skhattab, chatree, melhem, mosse, znati}@cs.pitt.edu

Abstract—**We propose a framework based on *proactive server roaming* to mitigate the effects of Denial-of-Service (DoS) attacks. The active server proactively changes its location within a pool of servers to defend against unpredictable and undetectable attacks. Only legitimate clients can follow the active server as it roams. We present algorithms that are secure, distributed, randomized, and adaptive for triggering the roaming and determining the next server to roam to. We propose some modifications to the state recovery process of existing TCP connection-migration schemes to suit roaming. Preliminary experiments in a FreeBSD network show that the overhead of server roaming is small, in terms of response time, in the absence of attacks. Further, during an attack, roaming significantly improves the response time.**

Index Terms — **Network Security, DoS Attack Mitigation**

I. Introduction

Legitimate network packets consume various kinds of shared resources, such as link bandwidth, router buffers, server memory, processing and operating system structures. Denial-of-Service (DoS) attacks inject maliciously-designed packets into the network to deplete some or all of these resources. *Server-oriented* DoS attacks, such as TCP SYN attacks [1], exploit vulnerabilities in operating systems and network protocol implementations to deplete server resources, whereas *link-oriented* DoS attacks flood critical links in the path of legitimate packets. The synchronized participation of many attacking nodes can achieve high illegitimate packet rate in a Distributed Denial-of-Service (DDoS) attack [2].

Our goal is to design and implement a framework for mitigating the effects of both types of DoS attacks, server-oriented and link-oriented. In this paper, we consider *private* services with *offline* subscription. We first give an overview of our integrated framework and then focus on one of its building blocks, namely proactive server roaming.

Our framework employs three lines of defense. The first line of defense is the utilization of Quality-of-Service (QoS) techniques within the network to isolate and protect traffic from/to subscribed clients from other traffic. VIPnet [3] adopts a similar approach. The second line of defense is *server roaming*, by which we mean the migration of the service from one server to another, allowing the old server to clean its state and perform source-code downloading from a secure read-only medium to defend against possible Trojan horses [4]. Finally, servers utilize resource management schemes, such as **netnice** [5], as a third line of defense.

In server roaming, the active server[1] changes its location within a pool of servers, either reactively in response to the detection of an attack, or proactively to defend against unpredictable and undetectable attacks. Only legitimate clients can follow the server as it roams. We assume that attacks and intrusions occur and it is not always possible to detect such malicious activities. The proactive nature allows our scheme to tolerate undetected attacks, whereas the reactive component allows the scheme to benefit from current advances in intrusion detection systems (IDSs) [6], [7].

The main challenge facing DoS defense systems is to *accurately* and *efficiently* distinguish between legitimate and illegitimate packets and to drop only illegitimate packets. Attacks which employ legitimate-like packets, such as reflector attacks [8] and insider attacks, complicate accurate detection. Also, efficient filtering implementations at line-speed are not easy to design because of IP spoofing, for instance. A more complex cryptography-based authentication scheme can become a DoS attack target by overwhelming it with many illegitimate authentication requests.

We envision our server roaming mechanism not only as a DoS defense by itself but also as a building block in a larger integrated DoS defense system. Server roaming helps to define accurate and efficient packet filters. Except for a small transitional interval (§IV-C), a non-active server can just assume that all packets destined for it are illegitimate and can drop them. A firewall at each server's entry point performs this adaptive filtering. Also, receiving a stream of *pure* attack packets provides on-line and accurate training data for IDSs and potentially easier and more accurate attack *traceback* [9].

Another advantage of server roaming is that it can detect legitimate but misbehaving clients that are potential insider attackers. A legitimate client violating its QoS contract with the service is provided a second chance after the service roams. If this client persists in violating its contract, it loses its status as a legitimate client because it is either faulty, malicious, or

The authors were supported in part by NSF under grant ANI-0087609.

[1] The scheme can be extended to allow for k out of N servers to be active at the same time to achieve load balancing.

has already been infiltrated by a malicious attacker.

In this paper, we propose a secure and adaptive framework for proactive server roaming. Each time the service roams, the roaming time and the address of the new server are kept secret except from legitimate clients. The maximum duration between successive roaming instances is a parameter of the scheme and is adaptively changed to strike a balance between the roaming overhead and the current threat level. Also, our framework allows for hierarchical trust levels among legitimate clients. The subscription duration for each client, during which the client can track the roaming server, is controlled to reflect the client's trust level.

The remainder of the paper is organized as follows: In Section II, we discuss related DoS defenses. In Section III, we present the system model and our main assumptions. We present our proactive roaming trigger mechanism and discuss how to migrate active connections in Section IV. In Section V, we present performance measurements of the proactive server roaming mechanism via a prototype implementation. Section VI concludes the paper.

II. Related Work

IP hopping [10] protects a *public* server, whose clients use DNS to look up its IP address. Our scheme protects private services; that is, legitimate clients know the IP address of the server directly. In IP hopping, the server changes its IP address after detecting it is under attack without changing its location. All packets destined to the old IP address are filtered at the network perimeter by a firewall. To avoid continuous server reconfiguration, a NAT (Network Address Translation) gateway can be used. IP hopping can be used both reactively and proactively. One limitation of this mechanism is that during the period of time in which the DNS entry of the old IP address is cached, all legitimate client requests are filtered out. Also, IP hopping does not block a persistent attacker which looks up the new IP address using DNS.

By physically moving the service from one server to another and cleaning the state of the old server, our scheme avoids the limitations of logical roaming schemes, such as IP hopping, which keep the server vulnerable to malicious state entries possibly implanted in the server during the attack. Moreover, illegitimate packets will still go to the same network after the address is changed, potentially congesting upstream routers.

Both the *TCP-Migrate* [11] and the *Migratory-TCP* [12] provide a framework for moving one end point of a live TCP connection from one location and reincarnating it at another location having a different IP address and/or a different port number. These mechanisms are used for mobility support and fault, or attack, tolerance. Our scheme builds on these mechanisms by providing a secure framework for issuing the roaming trigger both proactively and reactively. We also use them as a vehicle for our server roaming scheme.

III. System Model

We assume a pool of N homogeneous servers, geographically dispersed to minimize path intersections from clients to servers. Each server is connected to the network through a Firewall. Servers and firewalls employ *threshold cryptography* [13], *proactive secret sharing* [14], and *secure group communication* (e.g. [15]). We assume loose clock synchronization among servers (i.e., the clock shift is bounded by a constant, δ) and tight clock synchronization between each server and its firewall. We envision the protected *service* as a private, generic TCP-based service which can be replicated. *Legitimate clients* are subscribed to the service and can establish a secure channel[2] with any server and firewall. Each legitimate client has a QoS contract with the service to allocate a certain amount of resources to the client's requests.

We are defending against a malicious adversary which can be passive, active or both. A passive adversary can eavesdrop on client-to-server communication and apply cryptanalysis techniques. An active adversary can impersonate a legitimate client, obtain access to a legitimate client, inject malicious messages to servers and/or legitimate clients, and/or tamper with legitimate messages. The adversary can use some or all of these actions to launch a DoS attack, server-oriented or link-oriented. The attack can be launched from a large number of compromised machines (*zombies* [2]) which can likely be infiltrated client machines.

IV. Proactive Server Roaming

In this section, we present our secure, adaptive and time-triggered proactive roaming scheme and discuss the mechanisms we use for migrating active connections. We start by presenting our secure, distributed and randomized algorithms, based on *backward hash chains,* for the calculation of roaming times and locations of roaming destinations, allowing for an undetectable proactive roaming behavior.

A. Roaming Trigger

Our scheme utilizes backward hash chains, where members of the chain are used in the reverse direction of their generation. Hash chains are generated using one-way hash functions [17]. The lightweight computational overhead of hash functions allows for a simple and efficient implementation, and is suitable for use in mobile devices with computational and power constraints. The backward usage of hash chains is inspired by the work done in [16]. Backward hash chains provide forward secrecy which allows for the desired property of flexible rejection of misbehaving legitimate clients without the need to update the secrets. In the description that follows, H, H' and H'' are public one-way hash functions.

Figure 1 illustrates our scheme. We divide the service time into *epochs*. At the end of each epoch, the service migrates to a new server in the server pool. We generate a *long* hash chain using a one-way hash function $H(\cdot)$ and use each key K_i in the chain to calculate both the length R_i of the corresponding epoch E_i and the location of the active

[2] By secure channel we mean the provision of Confidentiality, Authentication, Integrity, and Message Freshness (immunity to message replay attacks) [16].

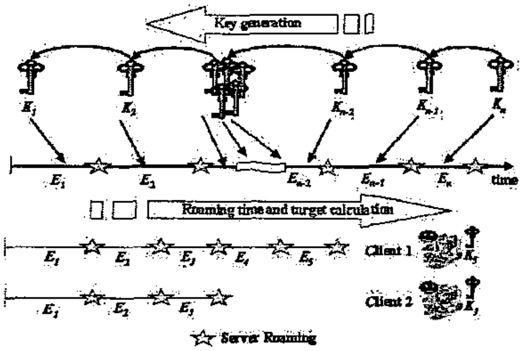


Fig. 1. Calculation of roaming times and targets using bakward hash chains

server S_i during E_i as follows: $R_i = \text{MSB}_m(H'(K_i))$ and $S_i = \textit{servers}[\text{MSB}_{logN}(H''(K_i))]$, where $\text{MSB}_j(x)$ are the j most significant bits of x, 2^m represents an upper bound on epoch length, N is the number of servers, and the array *servers* is a list of $\langle$IP address, TCP port$\rangle$ pairs of available servers in the server pool. Each server and firewall keeps track of the current service epoch.

During the *initialization phase*, a *group key* is generated and maintained using threshold cryptography [13] and proactive secret sharing [14] techniques. This group key is used for establishing secure group communication. The value of the last key in the hash chain, K_n, is randomly generated and made available to all the servers and firewalls in secure tamper-resistant media[3].

Offline registration involves the generation of a shared secret key, K_C, for each subscribed client to create a secure channel with each client. To reduce the risk of revealing K_C, the value of K_C is not used directly in encrypting the messages between the service and the client; instead, keys generated from K_C by means of a one-way hash function are used. According to each client's trust level and/or other policies, a key K_t from the hash chain is assigned to each client, with a varying value of t, allowing the client to track the service up to and including epoch E_t. The larger the value of t is, the more time the client can follow the roaming service. For instance, in Figure 1, client 2 is assigned K_3 and can thus track the service until epoch E_3, whereas client 1 can follow the service until E_5. Each client receives both the shared secret key and the value of K_t in a secure tamper-resistant medium. Clients also receive the *servers* list encrypted with K_C. The service registers the IP address of each subscribed legitimate client, a TCP port number for roaming update messages as will be described in the next paragraph, the shared secret key K_C, and the index t of the roaming key assigned to each client. This legitimate client list is made accessible to all servers. To ensure its integrity and confidentiality, the list is encrypted with the group key after appending a Message Authentication Code (MAC) [18].

Our mechanism defines *roaming update* messages to serve three purposes: (1) to allow for the change of the epoch length's upper bound, 2^m (to reflect the current threat level for example); (2) to keep the legitimate clients aware of the current service epoch; and (3) to provide a means for *on-line* extension of subscription durations of trusted clients; that is, to provide trusted clients with newer key values allowing them to stay connected to the service for a longer time.

The service uses the secure channels established with the clients to send *periodic* roaming update messages to currently subscribed clients at the roaming port set during offline registration. The reasons we do not send a roaming update message for each single roaming instance include: higher communication overhead, higher probability of a client missing a roaming instance due to an adversary blocking the update message, and more detectable roaming times for an eavesdropper on the update messages. It should be noted that roaming update messages caused by a change in the value of m are sent to all servers and firewalls as well.

A roaming update message contains one or more of the following items: (1) m, ℓ: starting from epoch E_ℓ the new value of m is used for calculating epoch lengths, where $\ell > r$ and E_r is the current epoch; (2) r, j, N_j: r is the current epoch of the service and N_j is the time until the start of epoch E_j, where $j \leq t$ and t is the index of the most recent roaming key granted to the client receiving the message; and (3) t, K_t: K_t is a new key from the service hash chain.

A legitimate client uses the values r, j and N_j received in a roaming update message to keep track of the current service epoch. Each client has a local variable, y, indicating the current service epoch. The client records the time it receives the roaming update message, T_u, and sets y to be equal to r. The client computes the remaining time N_y in the current epoch E_y as follows: $N_{v-1} = N_v - \text{MSB}_m(H'(K_{v-1}))$ for v starting from j down to $y + 1$.

Our scheme authenticates all roaming messages to ensure that no malicious adversary can inject false trigger messages to corrupt the roaming mechanism or to cause the legitimate clients to roam to a non-participating server, causing client DoS and, even worse, causing the clients to participate in a coordinated DDoS attack against the false roaming target.

B. Service Migration

We use TCP connection migration [11], [12] as a vehicle for our server roaming scheme. In this section, we define some requirements for the connection migration process in order to be able to defend against DoS attacks. We also propose and discuss some slight modifications to the current TCP connection migration mechanisms to suit these requirements.

In TCP connection migration schemes, both the application and TCP states should be recovered at the new server. A TCP connection migration scheme suitable for our roaming mechanism should assume very little or no dependence at all on the old server in the state recovery process, as it can be

[3] We are investigating using threshold pseudo-random generation techniques [13] to generate and maintain K_n.

already out-of-service due to the DoS attack. It also should be lightweight (i.e., with as small an overhead as possible).

We use periodic state updates instead of lazy, on-demand updates because there is no guarantee on the operating state of the old server to be alive and available for state requests. We engage the clients in the state recovery process, instead of using periodic server-to-server updates, due to two main reasons. First, server-to-server updates are expensive (e.g., we would need to update all the servers in order not to reveal the identity of the next roaming target). Second, the state of the old server might already be faulty, malicious, and/or overloaded, and migrating it completely would render the new server vulnerable as well. Pushing the state to the clients filters out malicious state entries at the new server because *only* subscribed legitimate clients know the address of the next roaming server and the roaming time and only these clients will be able to create state entries for their connections at the new server.

We propose to use the clients for storing periodic state checkpoints of both TCP and per-connection application state of the server. We use the same period for all connections, though starting from the time each connection was established (in other words, we want to phase the checkpoints so that to reduce the probability that a large number of state update messages are sent exactly at the same time).

State update messages are sent using the client's secure channel and to the same port as normal traffic to avoid the possibility of an attacker faking erroneous state updates and/or trying to block state update packets.

C. Discussion

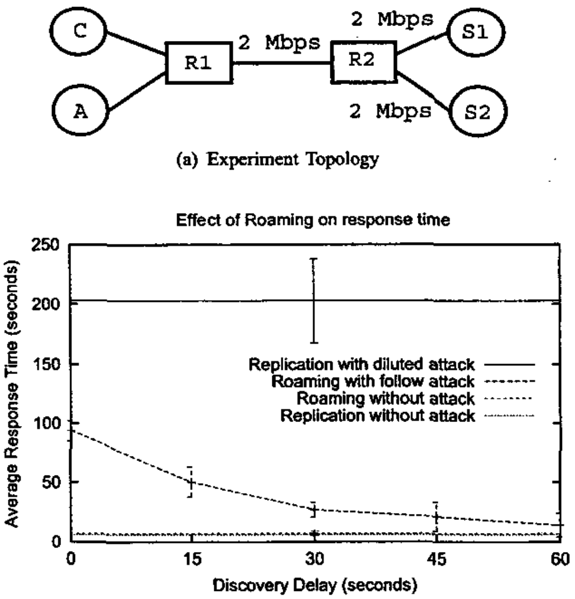
a) Transitional Interval: Because of the loose clock synchronization, each epoch should start earlier by δ at the new server and its firewall and end later by $\delta + \gamma$ in the old server and its firewall, where γ is the estimated communication delay in the network. This transitional interval constitutes a window of vulnerability during which firewalls accept all (illegitimate and legitimate) traffic, reducing its attack detection and filtering ability. Also, all legitimate traffic delayed in the network for more than γ will be considered illegitimate and can cause false positives in the attack detection process.

b) Analysis of Roaming Effectiveness: An adversary, having no knowledge of the K_i values, will be able to guess both the starting time of an epoch and the address of the epoch's active server with probability $\frac{1}{2^m N}$. The adversary must be able to guess both values correctly and continuously to launch an effective attack. An adversary can possibly choose a subset of M servers and either attack them continuously or change the attacked subset from time to time. Until the attack is detected and stopped, the effectiveness of the attack is reduced by the facts that the attack rate is diluted among the M servers and the service spends approximately $\frac{N-M}{N}$ of the time out of the attacked subset.

c) Advantages over Service Replication: Service *replication* can diffuse the load of link-oriented DoS attacks over a number of simultaneous replicas. However, replication is not DoS attack-tolerant in the same degree as it is fault-tolerant. For instance, replication cannot tolerate large-volume attacks that can clog all the replicas. Even with a smaller number of packets, server-oriented attacks can exploit vulnerabilities in all the replicas and knock them down almost simultaneously. We claim that our server roaming mechanism adds a layer of DoS attack-tolerance above plain replication. In the large-volume attack case, while the attack is building up at the active server, filters are being deployed upstream from each non-active server, potentially reaching and throttling the attack at its sources. For the server-oriented attack scenario, non-active servers do not accept the attack packets and thus stay unharmed. Moreover, they can use the attacking packets to train their IDS [6] and/or to deploy filters in the same manner as in the large-volume attack case.

V. Experimental Results

We developed an application-layer prototype in Linux for our mechanism. The prototype is a TCP disk-less file transfer service. We implemented a simplified version of the proactive roaming mechanism with fixed-length roaming epochs. Each client holds the amount of data received so far. Upon receiving a roaming trigger, each client drops current connections and establishes new ones with the new server to resume the transfer. When an active server receives a roaming trigger, it drops all its connections. A server does not accept incoming connections unless it is the current active server.



(a) Experiment Topology

(b) Response time with and without attack

Fig. 2. Experiment using an application-layer prototype

We conducted several experiments in a FreeBSD network and show here a proof-of-concept: Figure 2(a) shows the topology used in the experiment. Machine C issues 20 legitimate client requests with inter-arrival time from Poisson distribution with a mean of 8 seconds. Machine A, representing an attacker, issues file requests with a mean of 1 second for a duration of 5 minutes. Each request is for a file of size 1 MBytes, taking about 4 seconds to service in an empty network. Servers do not distinguish legitimate requests from attacking ones.

We modeled two types of attacks, *diluted* and *follow* attacks. In diluted attacks, the attack load is divided equally between the two servers, S1 and S2. A follow attacker directs the full attack load to the current active server. However, every time the active server roams, it takes some *discovery delay* time for the attacker to discover the server's new location. We test our scheme against discovery delay values of 0, 15, 30, 45, and 60 seconds. We use a fixed epoch length of 60 seconds. So, with 60 seconds discovery delay, the attacker always attacks the wrong server, whereas a discovery delay of 0 seconds represents an attacker who follows the exact server roaming behavior. We claim that the latter case happens with a low probability. We use follow attacks as an example of attacks that can target our server roaming scheme specifically.

We compare our roaming scheme with service replication, where legitimate client load is distributed uniformly between the two servers. The bottom two lines in Figure 2(b) represent the average response time in the case of no attack for replication and roaming. In an attack-free environment, roaming causes an increase of about 14% in average response time.

In the case of replication with diluted attack, each server uses its full 2 Mbps bandwidth to service requests, both legitimate and illegitimate, causing 4 Mbps traffic load at the 2 Mbps link from R2 to R1. With roaming, the idle server drops all connections and does not accept any new ones, thus consuming no bandwidth and causing no congestion in R2. Congestion can only happen in the active server's output queue; a problem which can be solved using resource management inside the servers. Moreover, during discovery delay in follow attacks, no attack requests arrive at the active server and thus it suffers no congestion. As the discovery delay increases, the congestion-free time increases, leading to smaller response times as shown in the graph. Server roaming is better than replication even with no discovery delay. The reason is that, with the start of each roaming epoch, all the existing attacking flows are dropped. This creates a small time window for the new server to service legitimate requests until the attack builds up again.

VI. Conclusion

In this paper we presented a framework based on proactive server roaming to mitigate DoS attacks. The current active server proactively changes its location within a pool of servers. Only legitimate clients are able to track the moving server. We presented secure, distributed and randomized algorithms for the computation of roaming times and addresses of roaming targets. We discussed some modifications and issues related to the application of TCP connection migration schemes to our roaming mechanism. Through a preliminary evaluation, we showed that our roaming scheme has insignificant overhead in attack-free situations and can provide good response time in case of attacks. We believe that this work is a new idea that can help, by itself or in coordination with other DoS defenses, in mitigating the effects of DoS attacks. For future work, we will test our mechanism under more realistic attack models and network configurations and investigate how to apply our server roaming scheme to protect public services, such as DNS and web servers.

Acknowledgements

The authors would like to thank Michael Gualtieri and Miguel Abele for their contributions in building the Linux prototype and useful discussions.

References

[1] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on TCP," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 1997, pp. 208–223.

[2] J. Mirkovic, J. Martin, and P. Reiher, "A taxonomy of DDoS attacks and DDoS defense mechanisms," Computer Science Department, University of California, Los Angeles, Tech. Rep. 020018, 2002.

[3] J. Brustoloni, "Protecting electronic commerce from distributed denial-of-service attacks," in *Proceedings of the eleventh International World Wide Web conference*, May 2002.

[4] L. Zhou, F. Schneider, and R. van Renesse, "COCA: A secure distributed on-line certification authority," Department of Computer Science, Cornell University, Ithaca, NY USA, Tech. Rep., 2000.

[5] T. Okumura, D. Mossé, M. Minami, and O. Nakamura, "Operating system support for network control: a virtual network interface approach for end-host OSs," in *Proceedings of IWQoS*, May 2002.

[6] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Depart. of Computer Engineering, Chalmers University, Tech. Rep., 2000.

[7] J. Green, D. Marchette, S. Northcutt, and B. Ralph, "Analysis techniques for detecting coordinated attacks and probes," in *Proceedings of USENIX Workshop on Intrusion Detection and Network Monitoring*, April 1999.

[8] V. Paxson, "An analysis of using refbctors for distributed denial-of-service attacks," *ACM Computer Communications Review (CCR)*, vol. 31, no. 3, July 2001.

[9] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for IP traceback," in *Proceedings of IEEE Infocomm*, 2001.

[10] J. Jones, "Distributed denial of service attacks: Defenses, a special publication," Global Integrity, Tech. Rep., 2000.

[11] A. C. Snoeren, H. Balakrishnan, and M. F. Kaashoek, "The migrate approach to Internet mobility," in *Proc. of the Oxygen Student Workshop*, July 2001.

[12] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode, "Migratory TCP: Connection migration for service continuity in the Internet," in *Proceedings of The 22nd International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[13] Y. Desmedt, "Threshold cryptography," in *European Trans. on Telecommunications*, vol. 5, no. 4, July–August 1994, pp. 449–457.

[14] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *LNCS 963, Proc. Crypto'95*, vol. 5, no. 4. Springer Verlag, 1995, pp. 339–352.

[15] C. K. Wong, M. Gouda, and S. Lam, "Secure group communications using key graphs," in *Proceedings of SIGCOMM*, 1998.

[16] A. Perrig, R. Szewczyk, V. W. andD. Cullar, and J. D. Tygar, "SPINS: Security protocols for sensor networks," in *Proceedings of MOBICOM*, 2001.

[17] O. Goldreich, L. Levin, and N. Nisan, "On constructing 1-1 one-way functions," in *Proceedings of the Electronic Colloquium on Computational Complexity*, 1995.

[18] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," in *Journal of the ACM*, vol. 33, October 1986, pp. 792–807.