Honeypot Back-propagation for Mitigating Spoofing Distributed Denial-of-Service Attacks *

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

September 8, 2004
University of Pittsburgh
Department of Computer Science
Technical Report TR-04-111

Abstract

Any packet destined to a honeypot machine (that is, a decoy server machine) is most probably an attack packet. We propose honeypot back-propagation*, a scheme that traces attack packets received by honeypots back to their source(s) and stops the attack source(s). In the proposed scheme, a server acts as a honeypot for some periods of time (*honeypot epochs*) and automatically triggers a hop-by-hop traceback of possible attackers (through back-propagation) during each honeypot epoch. In order to make the honeypots even much harder to evade, the start and end times of each honeypot epoch are unpredictable to attackers. We also propose* progressive back-propagation*, in which the information gathered during a honeypot epoch is used in subsequent epochs, to handle low-rate attacks, such as on-off attacks with short bursts. We developed an analytical model to estimate the expected time to reach and stop an attack source in the case of continuous and on-off attacks. Through ns-2 simulations, we validate our analytical model and show the effectiveness of the proposed scheme, which is attributed to obtaining accurate attack signatures and acting promptly once an attack is detected.*

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

1. Introduction

Denial-of-Service (DoS) attacks [33] aim at preventing a server from providing service to its legitimate clients. Exploiting software vulnerabilities (e.g., [10]) of improperly configured Internet nodes, malicious attackers can remotely gain control of a large number of *zombie* machines, to stage a highly distributed denial-of-service (DDoS) attack [24]. A DDoS attack can severely degrade the performance of its victims by entirely consuming their resources (e.g., [12]), ultimately bringing them to a complete halt. The subverted zombie machines can also be used in a DDoS attack to inject packets into the network at a rate high enough to clog the victim's connection to the Internet. Packets used in DDoS attacks can have authentic or spoofed (forged) fields; the source address is the most commonly spoofed field in order to hide the attack sources. In this paper we focus on defending against DDoS attacks with spoofed source addresses.

A large body of research in DDoS defense, namely *traceback* schemes, has considered tracing spoofed attack packets back to their sources [8,13,32,35]. However, most traceback schemes assume the existence of an accurate attack signature that provides them with the ability to distinguish attack packets from legitimate ones. Even in cases where such attack signatures are provided using Intrusion Detection Systems (IDSs) (e.g., Snort [5] and Bro [26]), analysis of packet logs and audit trails provided by the IDS is tedious and often leads to postmortem actions; in some cases attack signatures are inaccurate, thereby leading to a high rate of false positives. Moreover, the effectiveness of signature-based detection decreases substantially for low-rate DoS at-

tacks (e.g., [12, 21]).

The inability of a DDoS defense to accurately distinguish attack packets from legitimate ones can cause severe collateral damages. First, the fear of penalizing legitimate traffic limits the aggressiveness of the actions the scheme can take against attack traffic. Second, inaccurate attack signatures cause the scheme to traceback to legitimate users, thereby negating the benefits of the traceback process.

The objective of this paper is to enhance the effectiveness of traceback schemes in identifying and stopping malicious attackers by eliminating the negative impact of inaccurate attack signatures. To achieve this goal, we propose a hop-by-hop traceback scheme, referred to as *honeypot back-propagation*, which effectively traces back and stops sources of attack streams without impacting the performance of legitimate traffic streams. The main idea of the scheme is based on the concept of roaming honeypots [20]. Honeypots [30] are physical or logical machines (or networks) that have proven to be successful [22] in detecting worm-infected hosts [36]. Roaming honeypots takes this concept further by allowing the honeypots to move continuously and unpredictably to attackers among a pool of server replicas; each server in the pool, in coordination with the legitimate clients and the remaining peer replicas so as not to cause overall service interruption, assumes the role of a honeypot for specific intervals of time, referred to as *honeypot epochs*. Roaming makes it difficult for attackers to identify active servers, thereby causing them to be trapped in the honeypots. The designated honeypot (i.e., the server during each honeypot epoch) initiates the traceback process by alerting routers on the path to each attack source to enable rate-limiting and input-debugging [37] on traffic destined to the designated honeypot. The outcome of this process causes the access router of the attack source to reduce the rate of traffic destined to the designated honeypot to *zero*, effectively stopping the attack source.

The proposed scheme has several advantages over other DoS defense schemes, such as ingress filtering [15] and Pushback [23]. Similar to these schemes, honeypot back-propagation requires widespread deployment and cooperation among ISPs; its payoff, however, is higher: The proposed scheme avoids the collateral damage that Pushback may cause in some attack scenarios (more details in Section 6). Furthermore, the filtering mechanism used by honeypot back-propagation relies on the victim's destination address. Consequently, the proposed scheme no longer suffers from the management constraints of ingress filtering (see Section 2 for more details).

With a very large number of attack sources participating in a DDoS attack, it becomes increasingly difficult to take fast enough actions against the attack sources (e.g., to disconnect them from their ISP networks) even if their true IP addresses are known. The reasons include: (1) the attack sources may belong to a large number of administrative domains and (2) it may not be possible to block the attack sources based on their ip addresses as these ip addresses can be dynamically and continuously (re-)assigned and can thus be allocated to legitimate machines later on. Usage of destination-based filtering makes the proposed scheme scale to a large number of attack sources and avoids blocking legitimate machines that get assigned ip addresses previously allocated to attack sources. We note that the destination-based filtering used in honeypot back-propagation does not block legitimate clients from accessing the service because legitimate clients send their packets to service front-ends (as will be discussed in Section 3), which in turn tunnel these legitimate packets to back-end servers.

The rest of the paper is as follows. In the next section we review some of the related DDoS defense mechanisms. Section 3 presents the service and attack models. In Section 4, we describe the honeypot back-propagation scheme along with the required functions a router should implement to support the proposed scheme. In Section 5, we develop an analytical model for both the proposed scheme and two DDoS attack types, continuous and on-off attacks. We utilize the model to derive expressions for the expected time required to stop an attack (assuming full network support). In Section 6, we describe our ns-2 [6] model of the honeypot back-propagation scheme, which is based on both the Pushback and the roaming honeypots [4] ns-2 modules. We validate our analytical results, study the effect of different attack and system parameters, and compare honeypot back-propagation scheme to Pushback. Section 7 discusses some deployment issues and Section 8 concludes the paper.

2. Related Work

Because the Internet has no mechanism to enforce the validity of each packet's source address, attackers try to forge this field in the attack packets in order to hide their locations; this is called *spoofing*. In this section, we review some of the related proposed solutions of the spoofing DDoS attack problem.

If widely deployed, ingress filtering [15] and IPsec [18] can prevent most spoofed packets. However, the management hassle and per-packet performance overhead are obstacles against widespread adoption of ingress filtering and IPsec, respectively. Also, ingress filtering cannot defend against spoofing addresses from inside the ISP. The honeypot back-propagation scheme requires light-weight per-packet filtering based on each packet's destination address and, moreover, this per-packet filtering is only during the honeypot epochs and only in the case of attacks. The maintenance of ingress filtering rules can represent an overhead especially with dynamic assignment of ip addresses to sub-

networks inside the ISP and mobile IP [28] support. In honeypot back-propagation, filtering is based on the destination address of the victim and thus, the proposed scheme does not suffer from these management hassles.

The traceback problem, that is, trying to determine the real source(s) of the attack traffic and the network paths (attack paths) from these sources to the victim, has recently received attention from the network research community as a countermeasure of spoofing DoS attacks.

The first approach to the traceback problem uses a router feature called input debugging [37], which given a packet at a router's output port determines on which input port the packet was received. In its manual instantiation, the process starts at the router next to the victim, where upstream router(s) sending the attack traffic are determined. The process is then recursively repeated until the sources of the attack are determined, a router with no input debugging is reached, or the boundary of a non-cooperative administrative domain is met. To automate this process and limit the costly input debugging process to a few routers, the CenterTrack [37] architecture proposes to route victim's traffic through an overlay network of edge routers supporting input debugging. Another example of this link testing approach is controlled flooding [9], in which packet floods are injected into the network at selected points and attack paths are detected based on the perturbation these packet floods induce in attack traffic.

The Pushback [17] defense adopts a hop-by-hop strategy to propagate an aggregate-based rate-limiting filter upstream from a congested router. Both the detection of misbehaving aggregates and the assigning of rate limits are done using the Aggregate-based Congestion Control (ACC) [23]. At each Pushback-enabled router, a rate-limiting session for an aggregate is setup and the rate limit of the aggregate is shared in a max-min fairness fashion among input ports on which traffic matching the aggregate signature is received. To estimate the arrival rate of each input port, a feature similar to input debugging is used to map each packet at the output queue to its corresponding input port. The Pushback protocol [16] defines four messages. Request messages are sent from a router to one of its upstream neighbors carrying one or more uniquely identified aggregate signatures with corresponding rate limits and resulting in the establishment of one or more rate-limiting sessions and possibly other request messages to propagate the rate-limit further upstream. Each rate-limiting session has an expiration time; during a session's lifetime, the router sends periodic refresh messages upstream to maintain the soft-state-based rate-limit sessions at upstream routers and/or to update their rate limit. Pushback-enabled routers estimate the arrival rates of rate-limited aggregates and collect estimates from their upstream routers sent in status messages. The origin of a rate-limiting session may decide to explicitly cancel it through a cancel message if for example the reported arrival rate for the aggregate is below some threshold. Pushback accepts explicit misbehaving aggregate signatures through special request messages. Honeypot back-propagation scheme allows for the realization of this feature in the Pushback framework; when a server takes the role of a honeypot, the server's destination address forms the malicious aggregate. Selective Pushback [27] and level-k max-min fairness [39] address two limitations of the Pushback framework, namely the requirement of contiguous deployment and the hop-by-hop application of max-min fairness, which can severely punish legitimate traffic sharing the same aggregate signature as misbehaving traffic (a.k.a poor traffic) (see Section 6 for more details), respectively.

The second approach to the traceback problem is to have the routers send partial path information to sources of packets they forward, so that the complete path attack packets traversed can be recovered. This approach is known as the packet marking approach [32]. The packet markings are designed in a such a way to be used by their recipient to construct the attack tree(s) (a tree formed by merging attack paths with the victim at the root), to be lightweight for the routers to create and send, and to be backward compatible and incrementally deployable. The lightweight requirement is satisfied by probabilistically selecting a small fraction of the packets to send partial path information for. To achieve backward compatibility, either separate messages are sent carrying the path information [8], or the information is encoded in rarely used header fields (e.g. the IP identification field) [32]. To reduce false positives in the case of a large number of attack sources, enhance the running time of the path recovery algorithm, and prevent compromised routers from forging packet marks for other compromised routers, the advanced and authenticated marking schemes have been developed [35]. StackPi [29] is a deterministic packet marking scheme that allows the victim to locally filter attack packets based on their marking field.

Packet logging is the third approach to address the traceback problem. If routers log all the packets they forward, data mining techniques can be used to construct the attack tree [31]. However, this approach requires high storage requirements, especially in backbone routers. Hash-based traceback [34] utilize Bloom filters to address this problem and support tracing of a single IP packet back to its origin.

Some schemes facilitate the traceback problem by limiting possible attack sources or by providing accurate attack packets. By requiring that a small percentage of Internet routers filter packets for which the route from packet's source to packet's destination does not traverse the router, Distributed Packet Filtering [25] reduces the possible sources of an attack packet to a small number of networks. Roaming honeypots [20] divides the time into epochs and allows for a (continuously changing) sub-

set of k replicas in a pool of N replicas to be active during each epoch. The remaining $N - k$ replicas carry on the role of honeypots. The algorithm for selecting the k active servers during each epoch is pseudo-random to make it unpredictable to the attackers.

Finally, we consider the honeypot back-propagation scheme presented in this paper to be an example of an end-host-controlled [7] defense.

3. Models

We describe the context within which we envision honeypot back-propagation to operate, namely the service and attack models.

3.1. Service Model

We consider a service that is composed of front-ends, such as Web servers, and back-ends, such as database servers. Service front-ends are protected using massive replication (e.g., over a Content Distribution Network such as Akamai [1]), whereas back-end servers, which cannot tolerate the same level of replication because of tighter consistency constraints and higher costs, are replicated over a smaller number of replicas N. Thus, as depicted in Figure 1, the back-end servers are more vulnerable targets for flooding attacks. Front-ends distinguish legitimate traffic either through authentication in a subscription-based or private service or using client-legitimacy tests (e.g., [38]) in general. Determination of the starting and ending times of the honeypot epochs is done in coordination between back-ends and front-ends. By tunneling legitimate traffic to back-ends so that it does not reach a back-end server during one of its honeypot epochs, the front-ends convey the acquired traffic legitimacy information to the back-end servers to help protect the back-ends from DDoS flooding attacks. This model is similar to the service model in the roaming honeypots scheme [20] and carries some similarity with the SOS [19] architecture.

3.2. Attack Model

We consider n_a attack hosts (zombies) attacking a server S by sending spoofed packets destined to S or its subnetwork, aiming at clogging the link(s) connecting S's subnetwork to the Internet backbone. The effects of this attack are: (1) to block legitimate connection requests from reaching the server and (2) to degrade the throughput of both TCP flows from S into its clients as well as data flows from the clients into S. If TCP ACK packets (from clients to S) get dropped due to the attack, the throughput of TCP flows from S into its clients is degraded. Data packets from clients into

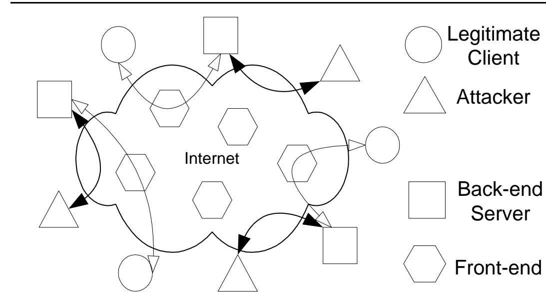

Figure 1. The service consists of a pool of back-end servers, a well-provisioned network of front-ends, and clients. Effective attackers send traffic directly to the servers.

S will be dropped as well; examples of such client-to-server data flows include clients uploading files into a server over TCP connections and nodes collecting data and streaming it, over UDP packets, into servers for processing and forwarding.

The access router of each attack host i $(0 \leq i < n_a)$ is h_i hops away from S. We consider two attack types, *continuous* and *on-off*. In the continuous attack, each attack host i continuously injects packets at a constant rate r_i. An attack host launching an on-off attack (depicted in Figure 2) alternates between sending packets at a rate r'_i packets per second during the on-burst (t_{on} seconds) and stopping for t_{off} seconds. Two interesting examples of the on-off attack are the TCP RTO attack [21] and the *follower attack*, in which the attack host is able to detect that a server is acting as a honeypot after a finite delay (follower delay) from the time the server assumes the role of a honeypot.

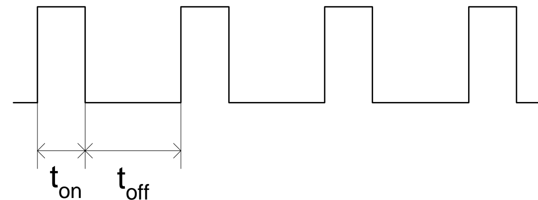

Figure 2. Parameters of the on-off DDoS attack.

Instead of sending attack packets destined to S, an attacker can send packets destined to addresses inside S's subnetwork. For machines (inside S's subnetwork) that accept traffic from the Internet, we assume that they are either servers that are protected using honeypot back-propagation

or machines that have a limit on their maximum incoming traffic rate (e.g., a DNS server inside S's subnetwork can have a limit on the rate of incoming DNS requests).

We note that the attack we are defending against is different from the *infrastructure attack*, in which attackers target their traffic into an infrastructure service (such as DNS [11]) or into routers along the paths to S from its front-ends. Applying honeypot back-propagation to defend a public infrastructure service is a subject of future work. Attacks on routers are beyond the scope of this paper and are difficult to be carried on anyway because of the high attack bandwidth required to clog a core router as compared to an access router [19], the possibility of routing around the attacked router(s), and the requirement of Internet topology information to select which intermediate routers to attack.

4. Honeypot Back-propagation

In this section we present the honeypot back-propagation scheme. We describe how we create time-windows in which a server receives a stream of pure attack packets and how this attack stream is used to reveal parts of the attack tree in order to trace the attack back to its sources. We also describe how honeypot back-propagation can be implemented progressively, that is, traceback information collected during one time-window are carried over to subsequent time-windows.

In coordination with service front-ends and other peer replicas (see Section 3), the server selects certain time intervals (honeypot epochs) during which it expects to receive no legitimate traffic; thus any packet destined to the server during one of these honeypot epochs is most probably an attack packet.

4.1. Basic Honeypot Back-propagation

As depicted in Figure 3(a), when a server S starts a honeypot epoch, it sends a *honeypot request* message to its next-hop access router(s). Upon reception of a honeypot request message, a router creates a honeypot session, during which the router applies a rate-limiting session with a small rate R_{min} (see Section 7 for a discussion of this parameter) at its output interface toward the node sending the honeypot request and on all packets destined to S. The router also starts input debugging at the output interface for packets destined to S. When a packet arrives at a router port x, if the packet's destination address is S, a honeypot request message is sent to the router connected to x (if it exists since this may be an access router) and the router stops logging packets from port x. Each router records the ports to which it has sent request messages. Figure 3(b) illustrates a snapshot of the scheme during a honeypot epoch. At the end of a honeypot epoch (Figure 3(c)), S sends a *honeypot cancel* message to its next-hop access router(s). When a router receives a honeypot cancel message, it sends a cancel message to all upstream routers to whom it has previously sent a honeypot request message. If the router has no upstream routers at all (i.e., an access router), it retains the honeypot session; otherwise it removes the session. If a long enough (see Section 5 for a discussion of how long is enough) attack stream is sent from the attack host(s) during a honeypot epoch, there will be one or more rate-limiting sessions at the access router(s) connected to the attack host(s) effectively stopping the attack, provided that all routers along the path from S to each attack host support the described functionality (we discuss incremental deployment issues in Section 7). After the administrators of (or any other appropriate policy at) the networks hosting the stopped attack hosts take appropriate action, a flush message is sent by the server to remove the rate-limiting sessions. Figure 11 provides a description pseudo-code of the basic scheme.

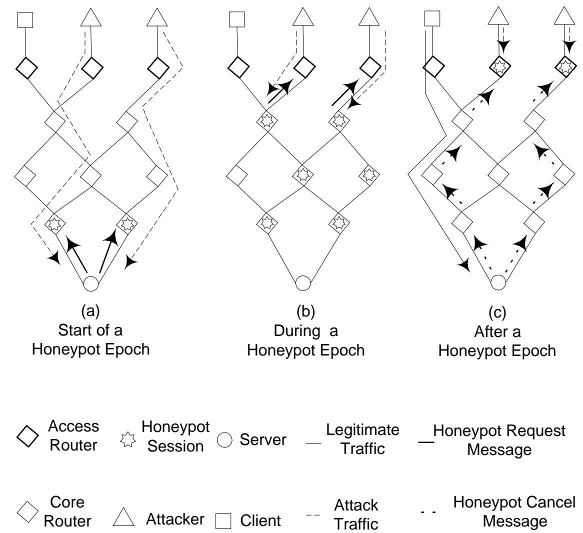

Figure 3. The operation of the basic honeypot back-propagation scheme at the start of, during, and at the end of each honeypot epoch.

4.2. Progressive Honeypot Back-propagation

In the basic honeypot back-propagation scheme, if an attack host does not send attack packets for a long enough time during a honeypot epoch, a honeypot request message will not reach the attack host's access router. If the attack host carries on an on-off attack with a short enough on-burst, the basic scheme will not be able to stop the attack. To

solve this problem, we propose an enhancement to the basic honeypot back-propagation scheme, in which the path to an attack host is discovered by accumulating information gathered during more than one honeypot epoch, as follows. In the following, let τ be the average time required for the honeypot request message to propagate one hop upstream and to setup a honeypot session as described in Section 4.1. S maintains a list of intermediate routers to keep track of the last non-access upstream routers at which no further propagation was possible at the last honeypot epoch. When a cancel message reaches a non-access router R (e.g., R_1 and R_2 in Figure 4(a)), the router checks if it has sent any requests upstream. If it has not done so (i.e., request propagation stopped at R), the router sends its identity R and a time stamp to the server S, which in turn calculates t_R, R's time distance (in seconds) from S and stores this information in the list of intermediate routers. At $t_R + \tau$ seconds before the next honeypot epoch, a request message (Figure 4(b)) is sent to each router R in the intermediate router list. This setting allows the back-propagation process to resume at the start of the honeypot epoch.

In order to maintain the intermediate router list and prevent explosion of its size, we apply the following two mechanisms. First, we remove a router R added to the list at honeypot epoch i from the list if R didn't send a message to S at the immediate next honeypot epoch $i + 1$. The fact that R didn't send a message at honeypot epoch $i + 1$ means that it either has propagated a honeypot request upstream (in which case it should be removed) or the report message was lost (a rare situation (see Section 7) in which propagation is restarted). Second, if S detects that an intermediate router R (e.g., R_1 in Figure 4) has not received any attack packet in ρ honeypot epochs, S removes R from the intermediate router list. S keeps a flag for each intermediate router entry R in the list to be able to detect if R did not send a message in the previous honeypot, in which case R should be removed. S also keeps a counter to detect when R reaches the threshold ρ. The pseudo-code for the progressive back-propagation is presented in Figure 12.

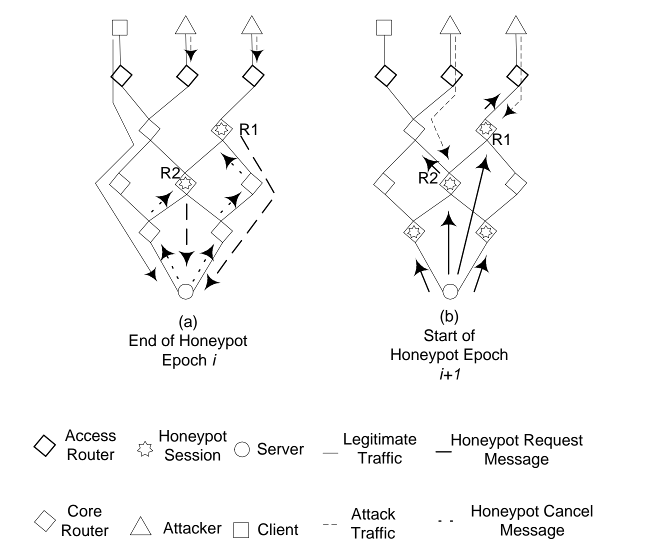

Figure 4. The operation of the progressive honeypot back-propagation scheme.

5. Analysis

In this section we study the expected time it takes to capture (stop) a DDoS attack host attacking a victim server using the honeypot back-propagation scheme. Let the CT_i (the *capture-time* of host i) be the time it takes to capture an attack host i. We derive conservative expressions for the average capture-time $E[CT_i]$ in the case of two types of attacks, continuous and on-off (see Section 3.2). In this analysis we assume all routers on the attack tree support the honeypot back-propagation scheme and we consider the progressive back-propagation scheme (if not stated otherwise).

We divide the time into epochs with constant length m. A server S decides to act as a honeypot during each epoch with a probability p (honeypot probability). Although this decision is done in a pseudo-random fashion in order for the front-ends to keep track of the honeypot epochs, the process appears as random to attackers.

5.1. Continuous Attack

For a continuous attack, it takes on average $\frac{1}{r_i}$ seconds for a honeypot session to receive an attack packet from attack host i. Also, it takes on average τ seconds to propagate a honeypot session one hop upstream. Thus, the average number of upstream hops honeypot request messages can reach during a honeypot epoch of length m seconds is $\geq max(1, \lfloor \frac{m}{\frac{1}{r_i}+\tau} \rfloor)$. The number of honeypot epochs required to propagate h_i hops (i.e., to reach the access router of attack host i) is $n_{hp_i} \leq \frac{h_i}{max(1, \lfloor \frac{m}{\frac{1}{r_i}+\tau} \rfloor)}$. In this setting, each epoch is a Bernoulli trial with a probability p of success (i.e., probability p of being a honeypot epoch). Thus, the average number of epochs needed for S to enter n_{hp_i} honeypot epochs is $\frac{n_{hp_i}}{p}$. Thus, the average time to capture a continuous DDoS attack host i is $m\frac{n_{hp_i}}{p}$.

In the progressive honeypot back-propagation scheme, honeypot request messages are sent before the start of each honeypot epoch in order for the honeypot sessions to start at the beginning of the honeypot epoch. Thus, the above analysis holds provided that at least one packet (on average) is received during each honeypot epoch, that is $m \geq \frac{1}{r_i}$ (for the basic scheme, $m \geq h_i(\frac{1}{r_i} + \tau)$ must hold). So, for con-

tinuous attacks,

$$E[CT_i] = m\frac{n_{hp_i}}{p} \leq \frac{m}{p}\frac{h_i}{max(1, \lfloor\frac{m}{\frac{1}{r_i}+\tau}\rfloor)}, m \geq \frac{1}{r_i} \quad (1)$$

Using the basic scheme and for some value of m, all attack hosts with rates $r_i < r^i_{max} = \frac{1}{\frac{m}{h_i}-\tau}$ cannot be captured. The total attack rate of these uncaptured attack hosts is at most $n_a\frac{1}{\frac{m}{h_{max}}-\tau}$, where h_{max} is the maximum hop-count distance from the server. With an estimate of the number of attack hosts $\hat{n}_a$, we can limit the effect of these uncaptured attackers by choosing $m \geq h_{max}(\frac{\hat{n}_a}{T}+\tau)$, where T is the maximum "unharmful" total attack rate.

5.2. On-Off Attack

For the on-off DDoS attack, we consider four cases depending on the relation of m with t_{on} and t_{off} (see Figure 2). As will be shown in the following analysis, the progress of honeypot back-propagation is guaranteed (on average) as long as: (1) when an on-burst overlaps a honeypot epoch, the minimum time of such overlapping is $\geq \frac{1}{r'_i}$ and (2) the threshold ρ (see Section 4.2) is large enough.

Case 1 ($\frac{1}{r'_i} \leq m \leq \frac{t_{on}}{2}$) In this case, for each on-burst of the attack, there exists at least one epoch e overlapped completely by the burst. We consider each on-burst as a Bernoulli trial with success probability p that epoch e is a honeypot epoch. The time between consecutive trials is $t_{on} + t_{off}$. Using the same analysis as with the continuous attack, the number of successful trials required to propagate h_i hops (i.e., to reach the access router of attack host i) is $n_{hp_i} \leq \frac{h_i}{max(1,\lfloor\frac{m}{\frac{1}{r'_i}+\tau}\rfloor}$). Thus, in this case, the average time to capture an on-off DDoS attack host i is

$$E[CT_i] = (t_{on}+t_{off})\frac{n_{hp_i}}{p} \leq \frac{(t_{on}+t_{off})}{p}\frac{h_i}{max(1, \lfloor\frac{m}{\frac{1}{r'_i}+\tau}\rfloor)}$$

Case 2 ($\frac{1}{r'_i} \leq \frac{t_{on}}{2} < m \leq t_{on}$) In this case, each on-burst of the attack will overlap with exactly one epoch e in $\frac{t_{on}}{2}$ seconds or more. Again, we consider each on-burst as a Bernoulli trial with success probability p that epoch e is a honeypot epoch. The time between consecutive trials is $t_{on}+t_{off}$. Using the same previous analysis, the number of successful trials required to propagate h_i hops (i.e., to reach the access router of attack host i) is $n_{hp_i} \leq \frac{h_i}{max(1,\lfloor\frac{\frac{t_{on}}{2}}{\frac{1}{r'_i}+\tau}\rfloor)}$.

Thus, the average time to capture an on-off DDoS attack host i in this case is

$$E[CT_i] = (t_{on}+t_{off})\frac{n_{hp_i}}{p} \leq \frac{(t_{on}+t_{off})}{p}\frac{h_i}{max(1, \lfloor\frac{\frac{t_{on}}{2}}{\frac{1}{r'_i}+\tau}\rfloor)} \quad (2)$$

Case 3 ($\frac{1}{r'_i} \leq t_{on} < m \leq (t_{on}+t_{off})$) The analysis in this case is the same as in Case 2. However, we note that the best attack strategy (in terms of increased attacker capture-time) fits in this case and sends just one packet during t_{on} and increases t_{off} as much as possible. In this case, because there is at least one attack packet received per epoch, $n_{hp_i} = h_i$. Thus, Equation 2 reduces to

$$E[CT_i] = h_i\frac{(t_{on}+t_{off})}{p}$$

The DDoS attack can be orchestrated using the the participating attack hosts to construct an aggregate on-off attack in such a way that each attack host is sending one packet every $t_{on} + t_{off}$ seconds. Although the proposed scheme will eventually stop such an attack, it may take a long time to do so. However, in order to orchestrate a DDoS on-off attack with on-burst t'_{on}, off-time t'_{off}, and on-burst rate r' using this strategy, the minimum number of attack hosts required is $r't'_{on}\lfloor\frac{(t_{on}+t_{off})}{(t'_{on}+t'_{off})}\rfloor$ ($r't'_{on}$ represents the number of packets per on-burst and $\lfloor\frac{(t_{on}+t_{off})}{(t'_{on}+t'_{off})}\rfloor$ denotes the number of on-bursts of the aggregate attack between two consecutive on-bursts of each participating attack host). So, as t_{off} increases, the number of attack hosts the attacker needs to compromise increases as well.

For Cases 1, 2 and 3, if ρ honeypot epochs pass without having a success (i.e., receiving at least one packet during a honeypot epoch), the last reached router will be removed from the list of intermediate routers (see Section 4.2) and the back-propagation process has to be restarted. Clearly, progress of the scheme is not guaranteed. However, the average number of epochs that pass until a success occurs is $\frac{(t_{on}+t_{off})}{mp}$. These epochs contain $\frac{(t_{on}+t_{off})}{mp^2}$ honeypot epochs on average. So, we set ρ to this value (after substituting $(t_{on} + t_{off})$ by a large enough value) to mitigate this problem.

Case 4 ($m > (t_{on} + t_{off})$) In this case, we let $T_m = t_{on} \cdot \lfloor\frac{m}{t_{on}+t_{off}}\rfloor$, so that each epoch overlaps with attack on-bursts for T_m or more seconds. The number of successful trials required to propagate h_i hops is $n_{hp_i} \leq \frac{h_i}{max(1,\lfloor\frac{T_m}{\frac{1}{r'_i}+\tau}\rfloor)}, T_m \geq \frac{1}{r'_i}$. Thus, the average time to capture an on-off DDoS attack host i in this case is

$$E[CT_i] = m\frac{n_{hp_i}}{p} \leq \frac{m}{p}\frac{h_i}{max(1, \lfloor\frac{t_{on}\lfloor\frac{m}{t_{on}+t_{off}}\rfloor}{\frac{1}{r'_i}+\tau}\rfloor)} \quad (3)$$

It should be noted that a continuous attack can be viewed as an on-off attack with $t_{off} = 0$, $t_{on} = \frac{m}{c} (c \in \mathbf{N})$, and $r'_i = r_i$. In this case, $T_m = m$ and substituting in Equation 3 with these values yields Equation 1.

In the above analysis of on-off attacks, we assumed the attack traffic is independent of the honeypot epochs.

An interesting attack type which depends on the honeypot epochs is the follower attack (see Section 3.2), in which the attack host stops sending traffic after the start of a honeypot epoch with a delay $d_{follow} > 0$. The average time to capture such an attack host is bounded above by $\frac{m}{p}\frac{h_i}{max(1,\lfloor\frac{d_{follow}}{\frac{1}{r'_i}+\tau}\rfloor)}, d_{follow} \geq \frac{1}{r'_i}$.

5.3. Roaming Honeypots

The roaming honeypots scheme [20] reduces to our honeypot back-propagation scheme with the following mapping: $p = \frac{N-k}{N}$, where N is the total number of servers and k is the number of concurrent active servers. We use the parameters suggested in [20] (shown in Table 1) for the experiments presented in this paper. From the above analysis, with $m = 10$ seconds, an attacker 5 hops away from the server can send with a rate up to 12Kbps without being captured by the basic honeypot back-propagation scheme, assuming a packet size of 1500 bytes and assuming $\tau = 1$ second (note that $m \geq h_i(\frac{1}{r_i} + \tau)$ must hold for the basic scheme). Thus, at least 125 attack hosts are needed to clog a 1.5Mbps link without being captured by the basic scheme. Also, with $p = 0.4$, and with all attack hosts sending at exactly 12Kbps, it takes about ($h\frac{m}{p} = h \cdot 25$) seconds on average to capture all the attackers, where h is the hop distance of the farthest (from the server) attack host's access router. However, an attack host at 10 hops or more from the server cannot be captured at all by the basic scheme. It can be captured though by the progressive back-propagation scheme in less than $h \cdot 25$ seconds on average as long as its attack rate is about 1.33Kbps or more for the considered parameter values.

Parameter(s)	Simulated Values
(N, k)	(5, 3)
m	10 sec

Table 1. Parameters of the Roaming Honeypot Scheme

6. Simulation Results

We have implemented the basic honeypot back-propagation scheme in ns-2 and conducted a number of experiments to study the effect of different attack and system parameters on the proposed scheme as compared to Pushback [23]. We start by describing the modifications we applied to the Pushback and roaming honeypots ns-2 modules to build the honeypot back-propagation scheme.

6.1. ns-2 Model

We modified the Pushback ns-2 module to disable the local aggregate-based congestion control [23]. We defined a new message type, honeypot request, and a new session type, honeypot session, with the behavior described in Section 4. We also modified the response of access routers to the cancel messages (see Section 4). We have also extended the roaming honeypots module to send honeypot request and cancel messages, to support roaming with legitimate UDP traffic, and to start and end each honeypot epoch a little bit later and earlier, respectively, to accommodate in-transit legitimate traffic and clock synchronization (see Section 7 for more details).

6.2. Model Validation

The first set of experiments validates the analysis presented in Section 5 for continuous attacks (validation of the analysis of the on-off attack is not shown due to page limit). We use a string topology with one server at one end and an attacker at the other end. We vary the epoch length m, the honeypot probability p and the hop distance between the attacker and the server (the length of the string topology) (h). We measure the average time (over 100 ns-2 runs) needed to capture the attacker (capture time) using the honeypot back-propagation scheme and plot this average together with Equation 1. The results (shown in Figure 5) enforce the fact that Equation 1 is an upper-bound for the average capture-time. It is important to note that if the condition

$$m \geq h_i(\frac{1}{r_i} + \tau), 0 \leq i < n_a \quad (4)$$

holds (for the simulation parameters we use, it holds), Equation 1 reduces to $E[CT_i] = \frac{m}{p}$, that is, the time to capture all continuous attack hosts is independent of any attack parameter.

6.3. Simulation Topology and Parameters

We have implemented a topology generator in ns-2 that, given a hop count and router degree distributions, generates a tree topology with a given number of leaves; this is instead of using a tree collected using traceroute measurements (e.g., from [2, 3]). Although a tree collected using Internet measurements is more realistic, we made a trade-off between flexibility and reality by using hop-count distributions roughly matching the distributions observed in measured trees (e.g., [2, 3]). In the simulation topology, we

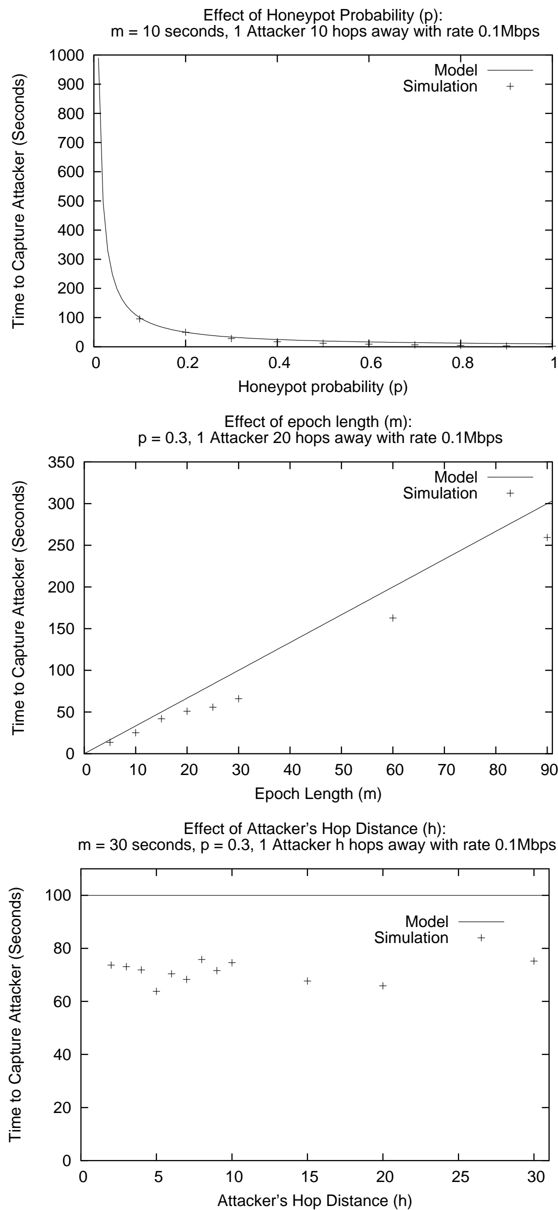

Figure 5. Model validation for the continuous attack.

model five servers behind a bottleneck link (1.0Mbps) representing the root of a tree generated using the hop count and router degree distributions shown in Figure 6. We did not try to create a power-law distribution [14] for the router-degree because as noted in [40], a power-law distribution is almost meaningless if the number of nodes is small (in the order of 100s), which is the case in the presented simulations. This tree represents the network paths from the tree leaves to the servers.

Links incident on leaf nodes have a 1Mbps capacity and 1ms propagation delay, whereas links incident on servers are 10Mbps in capacity and 1ms in propagation delay. All other links have a capacity of 10Mbps and a propagation delay of 10ms. Although these capacities and propagation delays are not real, their relative values roughly represent relations between access and core links and were used to expedite the simulations.

In this paper we report results with a network of 100 leaf nodes; we got similar results with different number of leaf nodes except that as the number of attack nodes increases, the required rate per attacker to clog the bottleneck link decreases, increasing the $\frac{1}{r_i}$ lower-bound on m (see Section 5). For example, with 500 attackers, it suffices for each attacker to send at a rate of 5Kbps to clog the 1Mbps bottleneck link. With a packet size of 1000 bytes, in the basic scheme, m has to be $\geq 2.6h_i$ (with $\tau = 1$ second) to reach an attacker h_i hops away from the server. So, to reach an attacker 10 hops away, m has to be at least 26 seconds.

We select legitimate clients and attack hosts from the leaves of the tree. Both legitimate clients and attackers send CBR traffic destined to the servers. In the results shown, we set the legitimate rate per node so that the total legitimate rate is about 90% of the bottleneck capacity (similar results were obtained with different legitimate loads). In the case of honeypot back-propagation experiments, at the start of each epoch, each legitimate client selects one of the three (see Section 5.3) active servers uniformly at random and directs its traffic into it. In the case of experiments with Pushback and with no defense, legitimate traffic is uniformly distributed over all five servers. For all experiments, each attack host picks a server among the five servers uniformly at random and keeps on attacking it. Each simulation run is for 1000 seconds. Legitimate traffic starts at time 0, while attack traffic is from 50 to 950 seconds. We measure the throughput of legitimate traffic as a percentage of the total bottleneck link capacity. Figure 7 shows an example of how the legitimate throughput changes with time during one simulation run.

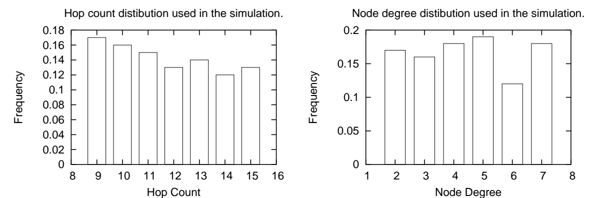

Figure 6. Hop count and node degree distributions for the simulated topology.

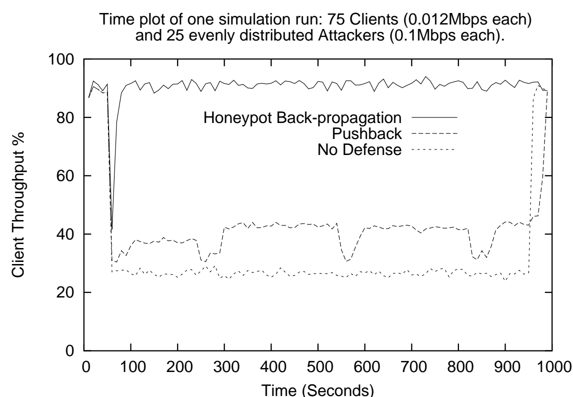

Figure 7. Time plot of one simulation run. Attack is between 50 and 950 seconds.

6.4. Effect of Attack Parameters

We study the effect of the location of attack nodes (in terms of hop distance from the victim server), the number of attackers, and the rate per attack host for continuous attacks. Because we are interested in the system behavior under attack, in the next figures we average the client throughput during the attack time (i.e., from 50 to 950 seconds) of each simulation run. Tables 1 and 2 summarize the studied parameters and the values we experiment with.

Parameter(s)	Simulated Values
Bottleneck Link Capacity	1.0Mbps
Total Number of Leaf Nodes	100
Total Legitimate Rate	90% of bottleneck capacity
Number of Attackers	25, 50, 75
Attacker Locations	Far, Close, Evenly Distributed
Rate Per Attacker	(0.01, 0.05, 0.1, 0.5) Mbps

Table 2. Simulation Parameters

6.4.1. Location of Attackers We consider three scenarios of attacker locations: (a) close attackers, where the attackers are assigned to the closest leaves to the victim in the topology tree, (b) far attackers, in which attackers are assigned to the furthest leaves from the victim, and (c) evenly distributed attackers, in which the location of attack nodes are selected uniformly at random over all leaf nodes. The legitimate clients occupy the remaining leaf nodes.

As depicted in Figure 8, as the locations of the attackers get closer to the victim, Pushback punishes legitimate traffic more. The reason for this behavior is that Pushback adopts a hop-by-hop max-min fairness allocation of the rate limit among upstream routers without taking into consideration the number of end hosts behind each upstream router. So, for example, the fair share of an end-host connected to a router with another two upstream routers is one third of the rate limit irrespective of the fact that the two routers may have many upstream end-hosts connected through them. The result of this behavior is that as attackers get closer to the victim, their fair share increases, reducing the residual rate for legitimate clients. For close attackers, Pushback is even worse than no defense at all because it even protects attack traffic.

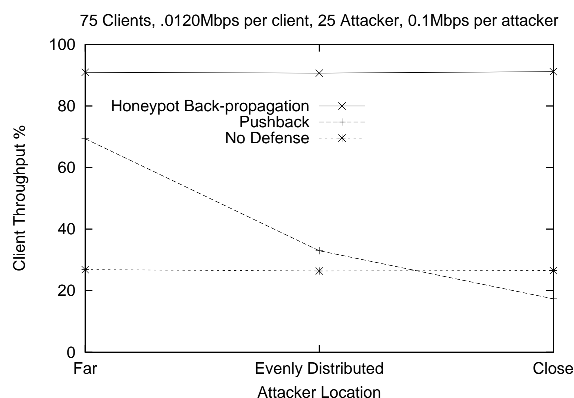

Figure 8. Effect of Attacker Locations.

6.4.2. Number of Attackers Figure 9 shows that honeypot back-propagation is independent of the number of attackers if condition 4 holds. For Pushback, as the number of close attackers increases, their aggregate attack rate, which is protected by Pushback (see Section 6.4.1), increases and the negative impact on the client throughput increases as well. For evenly distributed attackers, as the number of attackers increase, the number of attackers close to the victim increase leading to an increase of the protected attack rate. However, for far attackers (and close legitimate clients), client throughput is independent of the number of attackers, because in this case Pushback protects the legitimate traffic of the clients.

6.4.3. Rate Per Attacker As shown in Figure 10 (because condition 4 holds) honeypot back-propagation is independent of the rate per attacker in continuous attacks. For Pushback, in the case of far attackers, legitimate throughput is independent of the individual rate per attacker, because of Pushback protection of legitimate traffic of the close clients. For close attackers, as the attack rate for each close attacker increases, it can grab more of the rate limit, resulting in reducing client throughput.

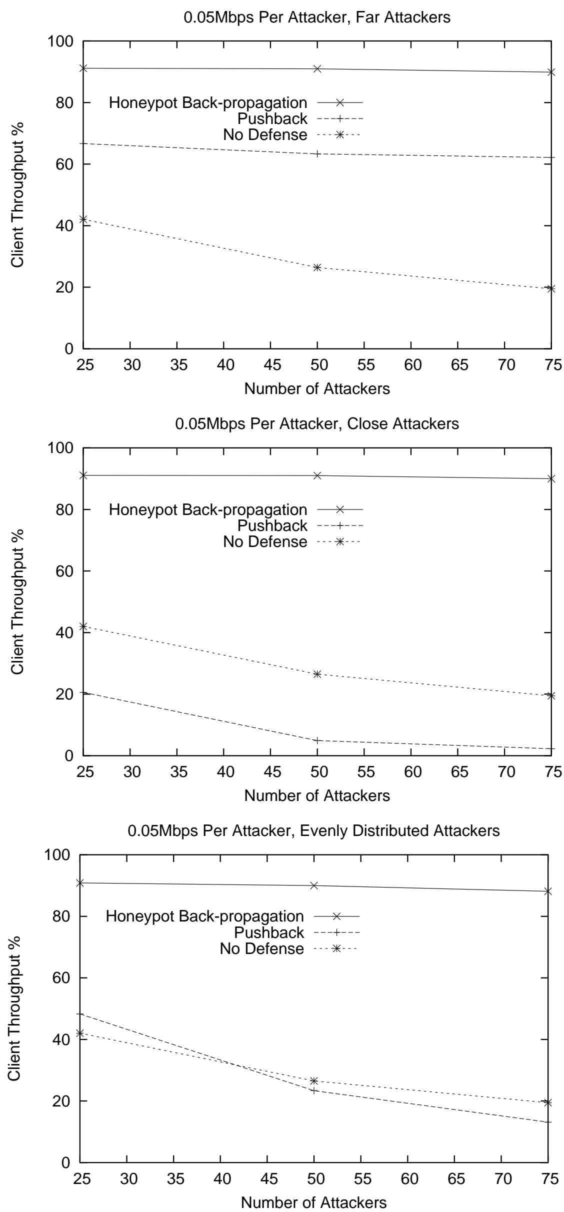

Figure 9. Effect of Number of Attackers for different attacker locations.

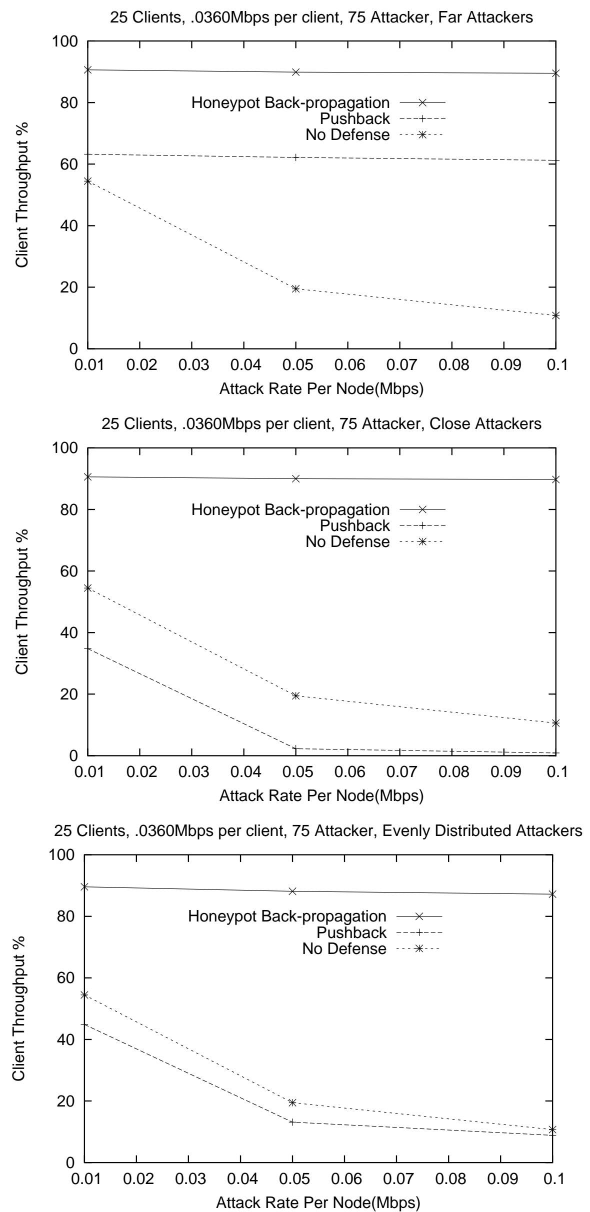

Figure 10. Effect of Attack Rate per Node for different attacker locations.

7. Implementation Issues

In this section we discuss some of the issues involved in implementing honeypot back-propagation in the Internet.

State Requirement Each router supporting honeypot back-propagation holds the state of the honeypot sessions it maintains during honeypot epochs. However, because honeypot requests propagate only in the case of DDoS attacks, if the server receives no (attack) packets during a honeypot epoch, no requests are sent upstream and no state is kept in the routers.

Message Security Because the proposed scheme can be exploited to launch a DoS attack by forging honeypot request messages (routers would block legitimate traffic), securing the messages in honeypot back-propagation is crucial to its deployment. Messages in the basic scheme are sent hop-by-hop and thus can be authenticated using the TTL field in the same way as in Pushback [23]. To defend against com-

promised routers and to secure the messages in the progressive back-propagation enhancement, we can use the lightweight time-released key chains [35]. Also, to minimize the chance of dropping one of the messages the proposed scheme sends, they can receive higher priority in router queues.

Incremental Deployment In order for honeypot back-propagation to reach an attack host and capture it, all routers along the path from the victim to the attack host should support the proposed scheme. Partial deployment is possible and results in the construction of partial attack trees. In progressive honeypot back-propagation, when a router R detects that its upstream router toward an attack source is not supporting honeypot back-propagation, R sends a message to S with this information. The action taken in this case of incomplete propagation is policy-based. Also, redirecting traffic destined to the server during each honeypot epoch into an overlay network (such as CenterTrack [37]) of supporting routers can enhance the incremental deployment feature of honeypot back-propagation.

False Positives If not configured appropriately, honeypot back-propagation can produce false positives in the following three cases: server-to-server control traffic, a legitimate client sharing the same access router with an attack host, and in-transit legitimate traffic. We handle each of these cases as follows. First, during honeypot epochs, a server may still need to receive some low-rate control traffic from peer servers or health monitors, for instance. To allow this traffic, we set the rate limit in honeypot sessions in all intermediate routers to some small rate $R_{min} > 0$. For all access routers except those connected to peer servers or preconfigured monitors, the rate limit is 0.

Second, in the case that an attacker and a legitimate client share the same access router, legitimate traffic will not be blocked by the honeypot session at the access router. This is because legitimate traffic is tunneled through service front-ends. If a front-end shares the same access router with an attacker, the front-end detects that its traffic is being dropped and reconfigurations take place in the front-end network to replace the blocked front-end.

Third, as common with most distributed systems, honeypot epochs have to be coordinated between the server and the front-ends, in order for the server to receive no legitimate traffic during honeypot epochs. We assume a loose clock synchronization between the server and its front-ends, with δ as the maximum clock shift. To handle loose clock synchronization and in-transit legitimate packets, each honeypot epoch at the server starts $\delta+\gamma$ after its scheduled time and ends $\delta + \gamma$ before its scheduled end, where γ is large enough to cover the propagation delay from front-ends to the server.

8. Conclusion

In this paper we presented honeypot back-propagation, a hop-by-hop traceback defense against DDoS attacks with spoofed source addresses. In coordination with its legitimate clients and other server replicas, a server enters honeypot epochs at times unpredictable to attackers. Honeypot back-propagation makes use of the stream of pure attack packets provided to the server during each honeypot epoch to reach and stop attack hosts without disrupting the operation of the replicated service. Using simple probabilistic analysis, we derive expressions for the expected time to stop an attacker. With appropriate selection of the system parameters, we show that it takes a reasonably small time to completely stop a DDoS attack given full network support. Through ns-2 simulations, we show the feasibility of the honeypot back-propagation scheme and show its effectiveness compared to the Pushback defense. Honeypot back-propagation allows for incremental deployment, and has limited overhead since routers only keep state during the limited honeypot epochs and only in the case of a spoofing attack.

References

[1] Akamai Corporation. http://www.akamai.com.

[2] Cooperative Association for Internet Data Analysis. http://www.caida.org/.

[3] Internet Mapping Project. http://research.lumeta.com/ches/map/.

[4] NetSec Group. http://www.cs.pitt.edu/NETSEC.

[5] Snort. http://www.snort.com.

[6] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[7] D. Adkins, K. Lakshminarayanan, A. Perrig, and I. Stoica. Taming IP Packet Flooding Attacks. In *Proceedings of Workshop on Hot Topics in Networks (HotNets-II)*, Nov. 2003.

[8] S. M. Bellovin, M. Leech, and T. Taylor. ICMP Traceback Messages. In *draft-ietf-itrace-01.txt, internet-draft, October 2001. Expired draft.*

[9] H. Burch and B. Cheswisk. Tracing Anonymous Packets to Their Approximate Source. In *14th Systems Administration Conference, LISA 2000*.

[10] C. C. C. A. CA-2003-04. MS-SQL Server Worm. http://www.cert.org/advisories/CA-2003-04.html, January 2003.

[11] CAIDA. Nameserver DoS Attack October 2002. http://www.caida.org/projects/dns-analysis/oct02dos.xml.

[12] S. A. Crosby and D. S. Wallach. Denial of Service via Algorithmic Complexity Attacks. In *Proceedings of the 12th USENIX Security Symposium*, August 2003.

[13] D. Dean, M. Franklin, and A. Stubblefi eld. An algebraic approach to IP traceback. *ACM Trans. Inf. Syst. Secur.*, 5(2):119–137, 2002.

[14] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-law Relationships of the Internet Topology. In *SIGCOMM*, pages 251–262, 1999.

[15] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofi ng. In *RFC 2827*, May 2001.

[16] S. Floyd, S. M. Bellovin, J. Ioannidis, K. Kompella, R. Manajan, and V. Paxson. Pushback Messages for Controlling Aggregates in the Network. In *draft-floyd-pushback-messages-00.txt, internet-draft, work in progress, July 2001. Expired draft*.

[17] J. Ioannidis and S. M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *Proceedings of Network and Distributed System Security Symposium*. The Internet Society, February 2002.

[18] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. In *IETF, RFC 2401*, November 1998.

[19] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *ACM SIGCOMM*, 2002.

[20] S. M. Khattab, C. Sangpachatanaruk, D. Mosse', R. Melhem, and T. Znati. Roaming Honeypots for Mitigating Service-level Denial-of-Service Attacks. In *Proceedings of ICDCS 2004*.

[21] A. Kuzmanovic and E. W. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks. (The Shrew vs. the Mice and Elephants). In *ACM SIGCOMM 2003*.

[22] J. Levine, R. LaBella, H. Owen, D. Contis, and B. Culver. The Use of Honeynets to Detect Exploited Systems Across Large Enterprise Networks. In *Proceedings of the 2002 IEEE Workshop on Information Assurance and Security*.

[23] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 62–73. ACM Press, 2002.

[24] J. Mirkovic, J. Martin, and P. Reiher. A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms. Technical Report 020018, Computer Science Department, University of California, Los Angeles, 2002.

[25] K. Park and H. Lee. On the effectiveness of route-based packet fi ltering for distributed DoS attack prevention in power-law internets. In *ACM SIGCOMM 2001*.

[26] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.

[27] T. Peng, C. Leckie, and K. Ramamohanarao. Defending against distributed denial of service attack using selective pushback. In *Proceedings of the Ninth IEEE International Conference on Telecommunications (ICT 2002)*.

[28] C. Perkins. IP Mobility Support. In *RFC 2002*, October 1996.

[29] A. Perrig, D. Song, and A. Yaar. StackPi: A New Defense Mechanism against IP Spoofi ng and DDoS Attacks. Technical Report CMU-CS-02-208, School of Computer Science, Carnegie Mellon, Pittsburgh, PA 15213, December 2002.

[30] T. H. Project. *Know Your Enemy*. Addison-Wisley, Indianapolis, IN, 2002.

[31] G. Sager. Security fun with OCxmon and cflowd. Intenet2 working group meeting, November 1998.

[32] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Pratical network support for IP traceback. In *ACM SIGCOMM 2000*.

[33] C. Shields. What do we mean by Network Denial of Service? In *Proceedings of the 2002 IEEE Workshop on Information Assurance and Security*.

[34] A. C. Snoeren. Hash-based IP traceback. In *ACM SIGCOMM*, pages 3–14, 2001.

[35] D. X. Song and A. Perrig. Advanced and Authenticated Marking Schemes for IP Traceback. In *Proceedings of IEEE Infocomm*, 2001.

[36] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, August 2002.

[37] R. Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[38] R. Thomas, B. Mark, T. Johnson, and J. Croall. NetBouncer: Client-legitimacy-based High-performance DDoS Filtering. In *Proceedings of DARPA Information Survivability Conference and Exposition - Volume I*, 2003.

[39] D. K. Y. Yau, J. C. S. Lui, and F. Liang. Defending Against Distributed Denial-of-service Attacks with Max-min Fair Server-centric Router Throttles. In *Proceedings of the IEEE International Workshop on Quality of Service (IWQoS)*, May 2002.

[40] E. Zegura. Thoughts on Router-level Topology Modeling. The End-to-end interest mailing list, 2001.

Appendix A: Pseudo-code for the basic and progressive honeypot back-propagation schemes

/*data-structures at the router*/
Marked[] /*Marked[i] $= 1$ iff a honeypot request message was sent on interface i*/

Procedure OnHoneypotEpochStart ()/*executed at the server*/
  Send a Honeypot Request message to next-hop access router(s) with the server's address
end

Procedure OnHoneypotEpochEnd ()/*executed at the server*/
  Send a Honeypot Cancel message to next-hop access router(s) with the server's address
end

Procedure OnHoneypotRequest (*server*)/*executed at each router*/
  Start a Honeypot Session and a Rate-limiting Session with rate R_{min} for a core router and 0 for an access router at the router's output interface toward *server*
  Enable input debugging on all packets destined to *server*
  Set Marked[i] to 0 for all the router's interfaces
end

Procedure OnHoneypotCancel (*server*)/*executed at each router*/
  If not an access router (near a client/attacker)
    Remove the Honeypot Session for *server*
  Send Honeypot Cancel message to all interfaces i for which Marked[i] = 1
end

Procedure OnPacket (*destination*)/*executed at each router*/
  Check to see if a Honeypot Session exists for *destination*
  If none exists, return
  Using input debugging, determine the input interface i on which the packet was received
  If (Marked[i] $==$ 0)
    Send a Honeypot Request message to next-hop router (if exists) at interface i
    Set Marked[i] to 1
    Exclude interface i from the input debugging process
  Determine whether or not to drop the packet according to the rate-limiting (rate limit set in Procedure OnHoneypotRequest())
end

Figure 11. Pseudo-code for the basic honeypot back-propagation scheme.

/*data-structures at the server*/
IntermediateRouters[] /*a list of intermediate routers in the attack tree at which back-propagation has stopped. Each list entry has:*/
R /*the router address*/, t_R /*an estimate of the time it takes to reach the intermediate router*/, rep /*the number of consecutive honeypot epochs the entry appeared in the list*/, and $current$ /*a flag to determine if the entry was received in the last honeypot epoch*/

/*data-structures at the router*/
Marked[] /*Marked[i] $= 1$ iff a honeypot request message was sent on interface i */
RequestSent /*a flag to determine whether the router has sent a request upstream or not*/

Procedure OnHoneypotEpochStart () /*executed at the server*/
  Remove all entries with a $current$ value of 0 or a rep value greater than some threshold ρ
  For each router R in the IntermediateRouters list, send a Honeypot Request message with the server's address to R t_R seconds before the epoch starts
  Send a Honeypot Request message to next-hop access router(s) with the server's address
end

Procedure OnHoneypotEpochEnd () is the same as in the basic scheme

Procedure OnHoneypotCancel (*server*)/*executed at each router*/
  If not an access router (near a client/attacker)
    If (RequestSent $==$ 0)
      Send an Intermediate Router Message to *server* with the router's address and a time stamp with the current time
    Remove the Honeypot Session for *server*
  Send Honeypot Cancel message to all interfaces i for which Marked[i] = 1
end

Procedure OnHoneypotRequest (*server*)/*executed at each router*/
  Start a Honeypot Session and a Rate-limiting Session with rate R_{min} for a core router and 0 for an access router at the router's output interface o toward *server*
  Enable input debugging on all packets destined to *server*
  Set Marked[i] to 0 for all the router's interfaces
  Set RequestSent to 0
end

Procedure OnIntermediateRouterMessage (R, *TimeStamp*)/*executed at the server*/
  If the Intermediate Router list has an entry for R
    increment the entry's rep field
  else
    add an entry for R
  set $current$ to 1
  set t_R to (Now - *TimeStamp*)
end

Procedure OnPacket (*destination*)/*executed at each router*/
  Check to see if a Honeypot Session exists for *destination*
  If none exists, return
  Using input debugging, determine the input interface i on which the packet was received
  If (Marked[i] $==$ 0)
    Send a Honeypot Request message to next-hop router (if exists) at interface i
    Set Marked[i] to 1
    Exclude interface i from the input debugging process
    Set RequestSent to 1
  Determine whether or not to drop the packet according to the rate-limiting (rate limit set in Procedure OnHoneypotRequest())
end

Figure 12. Pseudo-code for the progressive honeypot back-propagation scheme.