RAD: Reflector Attack Defense Using Message Authentication Codes

Erik Kline[†] Matt Beaumont-Gay[†] Jelena Mirkovic[⋆] Peter Reiher[†]
[†]Laboratory for Advanced Systems Research [⋆]Information Sciences Institute
UCLA Computer Science USC School of Engineering
{icebeast, mattb, reiher}@cs.ucla.edu mirkovic@isi.edu

***Abstract*—Reflector attacks are a variant of denial-of-service attacks that use unwitting, legitimate servers to flood a target. The attacker spoofs the target's address in legitimate service requests, such as TCP SYN packets. The servers, called "reflectors," reply to these requests, flooding the target.**

RAD is a novel defense against reflector attacks. It has two variants – locally-deployed (L-RAD) and core-deployed (C-RAD). Local RAD uses message authentication codes (MACs) to mark outgoing requests at their source, so the target of a reflector attack can differentiate between replies to legitimate and spoofed requests. MACs can be validated either at the target machine or on a gateway router at the target's network. Core RAD, which is deployed at the AS level, handles larger attacks that overwhelm L-RAD. The source AS marks each packet it sends with a hash message authentication code (HMAC) and core ASes filter packets that carry incorrect HMACs. C-RAD prevents reflector attacks by filtering spoofed requests, rather than filtering reflected replies.

We tested both variants using the DETER testbed by replaying backbone traces from the MAWI project archive in a congestion-responsive manner. Our tests show that Local RAD is better than the no-defense case, but gets overwhelmed when the attack exceeds the target's network capacity. Core-deployed RAD successfully handles attacks of all rates.

***Index Terms*—RAD, Reflector Attack, DoS, MAC, IP Spoofing**

I. Introduction

Denial-of-service (DoS) attacks have grown more popular and disruptive. Originally intended for revenge or bragging rights, they are now a tool for criminal extortion. DoS attacks can overwhelm a target's network connection with sheer volume of packets, or can exhaust some other resource at the target, such as SYN table entries. Network-based DoS attacks have two main attack vectors: direct and reflected [22]. In a direct attack, one or more machines directly attack the victim. In a reflected attack, the attacker uses intermediary machines to attack the victim. The attacker sends a request message to the reflector machine, spoofing the source address of the target. The reflector responds, sending the reply packet back to the victim. The attacker can send requests to multiple reflectors, flooding the target with replies. Reflector attacks are especially nefarious since they enlist the aid of unwitting machines to actually commit the attack.

Reflector attacks are difficult to detect at the reflector. Suppose the attacker controls 1,000 bots. Each bot sends one packet per second to ten different reflectors. The bot can rotate through a list of reflectors, sending to different reflectors every second. There are millions of possible reflectors, so each reflector may only get one request packet. The request appears legitimate, so the response will be sent. However, the target receives 10,000 response packets per second. A reflector attack is illustrated in Figure 1.

This research is supported by the National Science Foundation, grant CT-ISG 0716829.

For a reflector attack to work, the attacker(s) must be able to spoof the victim's IP address. The Spoofer project [32] investigates the question of how much spoofing is possible in the Internet by allowing volunteers to download test software and analyzing the results. Their results show that roughly 80% of probed networks filter randomly spoofed traffic. However, their study only covers around 1,500 autonomous systems (AS), about 3% of all ASes today. Furthermore, even if only 20% of all networks allow spoofing, they can still generate significant amounts of spoofed traffic. Analysis of backscatter traffic [20] shows that a significant number of DoS attacks use spoofing.

We propose the Reflector Attack Defense (RAD), consisting of a local defense and a core defense. Local RAD uses the tendency of protocols to repeat values from the request packet in the reply. Examples include the ID field in DNS requests and ICMP ECHO requests and the initial sequence number in a TCP SYN, which is incremented by one and repeated in the SYN/ACK. We use these repeated fields to carry a message authentication code (MAC). When a request is generated, the MAC is placed in the field. When a reply is received, the MAC is validated and invalid packets are discarded. Thus, a source can determine if reply packets truly correspond to sent requests. The MAC can be validated at the source's gateway or border machines, filtering attack traffic prior to disrupting the target machine.

However, if the attack completely overwhelms the target's bandwidth, Local RAD will also be overwhelmed. Core RAD addresses this problem by allowing an AS to mark all of its outbound IP traffic with a MAC. IP traffic from this source can then be verified in the core and spoofed traffic can be filtered. Unlike some other defenses, Core RAD provides ASes the capability of directly defending themselves from IP spoofing.

II. Local RAD

Local RAD (L-RAD) allows machines to validate whether reply messages correspond to request messages they generated,

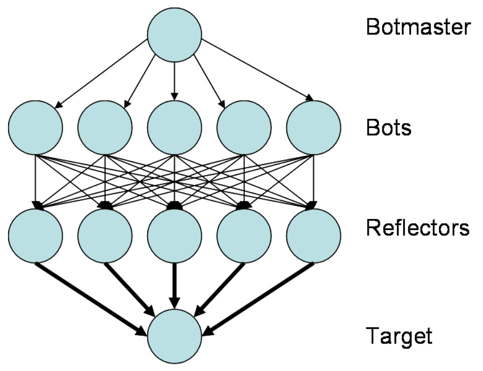


Fig. 1. Illustration of a Reflector Attack. The botmaster signals his bots to initiate the attack on the victim. The bots send out spoofed request messages with the victim's source address to the reflectors. The reflectors respond by sending replies to the victim, causing a flood-based DoS.

dropping those that do not. This can be accomplished in two ways. First, when a request is sent, state is kept in a table. When the reply is received, check for a matching request in the state table. This approach has a few drawbacks. We must generate and store the state table, which many existing protocols do already, e.g. TCP. However, lookup in a state table can be costly. During normal behavior, this cost is only incurred for legitimate replies. The cost comes during an attack, when the table is constantly searched for non-existent requests. The problem is worse if the victim responds to the false replies. For example, TCP sends a RST message if a unexpected SYN/ACK is received. During an attack, TCP will be generating a large number of RSTs, adversely affecting the target's normal outbound traffic.

We chose a stateless approach, so the response must contain information to tie it to the request. Luckily, most reflector message types carry a repeated value in both requests and replies. For example, ICMP ECHO packets have a 16-bit identifier field that is repeated in the ECHO REPLY. Similarly, the initial sequence number in a TCP SYN is incremented by one and repeated in the corresponding SYN/ACK. L-RAD generates a message authentication code (MAC) that can be placed in these repeated fields and be validated in the response. The MAC can also be validated at other locations, such as the gateway routers for the source's local network. These routers need to only keep minimal state themselves, but can dropping the attack packets prior to the source receiving them.

The L-RAD technique resembles SYN cookies [4], which also generate a hash using the source and destination IP addresses, ports and a counter. However, SYN cookies are designed to stop SYN floods, and thus their application is different. SYN cookies are placed in SYN/ACKs, while L-RAD MACs are placed in SYNs.

One benefit of L-RAD is that no new infrastructure is required. L-RAD can be deployed on only one machine, providing that machine with protection. Or it can be deployed to protect an entire network using the gateway router filtering method.

Local RAD has three components: MAC generation for request packets, MAC validation for reply packets, and packet filtering at gateway routers using the MAC.

A. Request MAC Generation

In L-RAD, every request message must carry a MAC that is difficult to forge without secret source information. To accomplish this, the source generates a random secret. To generate the MAC, we hash over the secret, some values from the request packet header, and a counter.

The MAC must be unique for each request, to prevent an attacker from reusing an intercepted MAC to generate reflector attacks to many different hosts. To accomplish this, we generate a 512-bit block consisting of source and destination IP addresses, as well as the ports. This method is still vulnerable to a replay of a captured request packet, so we also add a local counter to the block. Every time the counter increments, the hash generated for the same destination changes. This system also allows us to prevent collisions for the generated MACs. Since the MACs are used as identifications values and initial sequence numbers, they should be distinct from previous values sent to the same destination. Finally, we fill the rest of the block with the randomly generated secret, which is 384 bits. We then generate a SHA-1[1]digest and use the first 16 bits (for ICMP and DNS) or 32 bits (for TCP) as our MAC[2].

ICMP messages do not have ports. For them, we use only the source and destination IP addresses and the counter to calculate the MAC. We place this MAC in the 16-bit identification field and randomly pick a seed sequence number.

The counter value must be incremented frequently enough to encompass the round-trip time (RTT), but not so often as to create a large chance for collisions and a large vulnerability window. Since 90% of all TCP bytes transferred have a RTT of under 500 ms [11], and only 15% of connections have a median RTT greater than one second [1], we chose a counter increment time of two seconds. However, the counter increment time is configurable. In a particularly lossy environment, one may increase the counter increment time at the cost of security.

B. Reply Validation

Reply validation checks if the request and reply MAC match. When the reply is received, we use the packet header values and the counter to calculate a new MAC, which we compare to the reply MAC. If equal, we process the packet normally. If not, we drop the packet. Requests may be generated at the edge of the counter window resulting in an incremented counter and the dropping of legitimate replies. To solve this, we check the MAC with both the current and previous counter values. This slightly increases the window for which replayed packets will avoid filtering.

[1]SHA-1 is vulnerable to collision attacks [6]. In a collision attack, an attacker uses a valid MAC and an input to create another input that hashes to that MAC. In our system, if the attacker has a valid MAC, he can already attack us, without needing a second input. Also, a second input may be useless if it cannot be used to construct an attack packet.

[2]We used the chi-square randomness test to show that truncating the SHA-1 digest still results in uniformly distributed output.

C. Gateway Filtering

In L-RAD gateway filtering, the source generates the initial MAC, but its gateways or border routers validate replies. Gateway filtering has multiple benefits. First, reflector attacks can overwhelm the victim with pure packet volume. If we filter and drop the packets at the target, it may be too late. Filtering at the gateway takes the load off the target. Assuming the gateway has more processing power or bandwidth, the target can continue to operate normally without experiencing the brunt of an attack, while the gateway continues to filter. Second, if the target has multiple gateways, each gateway will only have to filter a portion of the source's traffic. In either case (larger bandwidth or multiple gateways), an attacker would have to generate more traffic to overwhelm the gateways than it would to overwhelm the target. Third, a gateway can defend multiple machines, rather than each machine fending for itself.

In a stateful gateway system, the gateway makes a table entry for each outgoing request packet. When a reply packet is received, it checks for a corresponding entry. However, this system has a few problems, beyond the basic table storage and lookup problems with stateful systems. First, a request and reply pair may take different routes, so one gateway might receive the request while another would receive the reply. This would require synchronization between the gateways. Aging table entries is also important. One could just remove an entry when the reply is received. But in TCP, if the ACK completing the three-way handshake is lost, the server will send a new SYN/ACK which would not have an entry in the gateway's table. Finally, this solution may not be applicable if the source wishes to be protected by its service provider. A service provider, while having more resources to protect the source, may not be willing to set up firewalls that may adversely affect others' traffic. They may be more willing to set up a stateless filtering system that only affects the source's traffic (and malicious traffic).

Therefore, we designed a stateless system. This system validates MACs in reply packets just as source-based filtering does, which requires that the gateways know the source's secret and counter value. All other validation information is present in the reply packet. We could distribute the secret using a simple reliable protocol to talk to each gateway, or we could broadcast the secret periodically. A gateway will only begin filtering once the secret is received. Finally, we could manually configure the secret on both the source and the gateways. We currently use the first option.

If the source believes that the secret is compromised, the secret is changed. Compromising the secret requires the attacker to determine 384 bits. Compromising a filtering node or using brute force are the only methods we have found for learning a secret. Brute force is not feasible, as it would take 2^{383} attempts. Since we believe filtering node compromise will be rare, the secret does not have to be changed frequently.

We also address counter synchronization. During secret distribution, we also inform the gateways of the initial counter value. The gateway sets its counter value and increments every counter interval (two seconds), as does the source. To synchronize, the source sends a beacon with its current counter value using UDP. When a request packet is to be sent, if more than a minute has passed since the last beacon, the source will send out a new beacon. Because these gateways are on the source's local network, the chances of losing the beacon are low. If the beacon is lost, the gateway should still be roughly in sync based on its old counter value.

With the secret and the current counter value, the gateway can filter incoming reply packets. Any packet with an invalid MAC is dropped, while all other packets are forwarded normally. Gateway filtering checks the MAC with the current, previous and next counter values, to handle minor desynchronizations arising from the use of a relatively large counter interval. Checking the next value also lessens the effects of losing a beacon since the valid counter window is longer.

III. Core RAD

No matter how well Local RAD performs, it will eventually be overwhelmed. In general, the best place to defend is closest to the attacker, so Core RAD (C-RAD) pushes the general RAD concept out into the core of the Internet. Once in the core, we have the capability to see the spoofed requests, not just the reflected replies, so we try to filter out the spoofed requests, alleviating the burden on both the target and the reflectors.

There are two major differences between L-RAD and C-RAD. First, it is not feasible for core routers to do deep packet inspection, which an L-RAD style scheme requires. Routers do not have the time to inspect a packet, determine its protocol, determine if it is a request, and validate the MAC. Therefore, a C-RAD source marks all its outbound IP traffic, and C-RAD filters validate all protected packets. Second, core routers cannot keep information for every host. Therefore, C-RAD marks at the AS level. An AS marks all its outbound traffic, and core routers only store information regarding the AS and its prefixes.

Core RAD has three major components. The first generates the MAC and marks packets at the source AS. The second validates the MAC on the core routers. The third synchronizes source ASes and core routers.

A. IP MAC Generation

C-RAD's MAC must be secure and quick to calculate. We use a Hash Message Authentication Code (HMAC). HMACs take a secret key as additional input, unlike L-RAD, which has a secret added directly to the data before running the hash algorithm. We chose the HMAC-SHA-1 hash algorithm for our implementation, but it can be changed easily.

The HMAC must be strong enough to prevent an attacker who has determined one valid HMAC from using it on many attack packets. To counter this, we hash over the packet payload, so one HMAC cannot be used for other packets. We cannot hash over the IP header, as fields in the header change per hop (such as TTL and checksum), invalidating the HMAC. However, if we leave it as is, an attacker can use one HMAC and its corresponding payload to attack many targets. To counter this, we include the source and destination IP addresses in the hash. If an attacker intercepts or determines a single valid HMAC, it can only be used for one payload to one reflector.

The HMAC must be stored in the IP header to allow for easy extraction (without inspection). Avoiding inspection also excludes IP options. Further, IP options are a bad candidate because they generally force a packet to take the "slow path." These restrictions only leave a handful of rarely used fields in the IP header. To provide maximum security, we chose the largest of these, the IP ID field.

The ID field is 16 bits, so the attacker only has to iterate over 2^{16} possible combinations to find a valid HMAC. However, since the HMAC is tied to the source and destination IP addresses, the attacker has difficulties validating if a guessed HMAC worked. The attacker could continuously iterate over the ID space, hoping the few packets that get through are enough to generate a DoS. In this case, the attacker would have to send out 128,000 packets per second to get a 2-packet-per-second flood. A large number of bots, each sending at this volume, could generate a useful attack; however, this attack would be easily detectable. Further, a reflector attack is less attractive with such a low yield, so an attacker would be likely to instead send spoofed packets directly from the bots to the target.

B. *IP MAC Validation*

When a core router receives a packet, it does a prefix lookup on its source IP address, which returns the secret key for that AS. The key, the payload and the IP addresses are used to generate the HMAC, which is compared to the packet's HMAC. If they do not match, the packet is dropped.

This process must occur at line speed for C-RAD to be effective. The bottleneck is HMAC generation. We tested our HMAC implementation on a Dual Xeon 3 GHz machine. It can handle about 100 Mb/s on 40 byte packets, and nearly 350 Mb/s on 1500 byte packets, because SHA-1 is more efficient for larger data sizes. Thus, the limiting factor is small packets, as is the case for core router forwarding.

100 Mb/s is not nearly fast enough. Current core routers forward packets at 10 Gb/s. While our software implementation may be slow, cryptographic hardware (as of 2004) can achieve 1 Gb/s [16]. Hardware is likely to continue to improve and achieve speeds of 10 Gb/s. Also, packets can be validated in parallel. If we use ten cryptographic components in parallel, we can achieve 10 Gb/s. This will, of course, increase per-packet delay, but will not reduce throughput. Note that C-RAD has no bearing on the forwarding lookup. If C-RAD can achieve 10 Gb/s throughput, we can run both processes in parallel.

Our scheme resembles Packet Passports [18]. Passports work by attaching a series of marks to a packet, each made by a secret shared by the source and one AS on the forwarding path. These passports are validated using an HMAC algorithm and the system also uses a Bloom filter to detect replay attacks. Packet Passports were able to obtain throughput on par to C-RAD.

C. *Router Synchronization*

Each source AS must distribute a key to the participating routers, and it must be able to change the key to limit replay attacks. However, we do not want to constantly bother routers with synchronization messages. It would be better if the routers could determine the next key without any additional information from the source after the initial message.

C-RAD achieves this by using reverse hash chains [10] to determine the key. When an AS wants to use C-RAD, it distributes the seed to the reverse hash chain and a timestamp. The router uses the seed to generate the reverse hash chain, using the first value as the key. After a predetermined time, the router moves on to the next element of the reverse hash chain.

We must change keys before an attacker can learn a valid HMAC. An attacker can easily iterate through all possible HMACs, but it is harder to determine if an HMAC is valid. One method of determining this is a probing attack based on Bellovin's work [3] and RocketFuel [29]. In this attack, the attacker sends a series of legitimate probe packets to the reflector. Some hosts sequentially chose the IP ID values to send in the response. If the value increments by more than one, traffic from another source arrived between the attacker's packets. To determine an HMAC, the attacker sends a legitimate probe, the spoofed packet, and then another legitimate probe. If the second probe only increments by one, the spoofed packet did not arrive. If it increments by more than one, the spoofed packet arrived, and thus contained the correct HMAC.

Many factors affect how long it takes this attack to succeed. For example, the attacker cannot know if the packet was filtered or lost unless he re-tests. Further, other traffic ("collisions") that arrives at the reflector between the attacker's probe packets will disrupt the measurement. If the attacker does not wait for one probe to complete before launching the next, he risks colliding with his own packets, or out-of-order packet delivery, which disrupt his measurements. The probe must travel through the core in order to test an HMAC, so the attacker cannot select a nearby reflector and must guarantee some minimum round-trip time.

The safe frequency for secret changing depends on how quickly an attacker can find a valid HMAC with reasonable probability. We make attacker-friendly assumptions: a round-trip time of 100 ms, 0% chance of collisions, 0% chance of packet losses, and no verification of success required. Because packets must travel through the core to probe the filters, 100 ms is liberal. These variables generate the following equation.

$$T = P_s \cdot 2^{16} \cdot RTT$$

P_s is the probability of success, RTT is the round-trip time, and T is the time to reach that probability. For example, given our assumptions and a P_s of 50%, it takes roughly 55 minutes

TABLE I
AVERAGE, MAXIMUM, MINIMUM AND MEDIAN CPU TIME TO ACCOMPLISH THE INDICATED TASKS. ALL TIMES ARE IN μS.

	Average	Maximum	Minimum	Median
SYN Generation	16.1474	45.8544	5.8607	14.2825
SYN/ACK Handling	9.2372	35.9910	3.5102	9.0955
RST Generation	0.8832	23.4252	0.1731	0.3782
SHA-1	1.4408	-	-	-

to achieve a 50% chance of guessing a valid HMAC. This gives the attacker only one HMAC, valid for one payload to one destination. To generate an actual attack, the attacker must run many probes in parallel to many reflectors, and then flood those reflectors with replayed traffic. Since the replay is the same packet per reflector, reflectors are more likely to detect the attack. This probing method is also not stealthy. A vigilant reflector could detect this method of probing and purposefully collide the probes or drop them.

Thus, a reasonable frequency of change is between 30 minutes to an hour. This allows for much looser synchronization than for L-RAD. If the routers are out of sync with a source for one second, only 0.06% of that source's packets will be lost, assuming a 30-minute window.

Still, it is important to keep the routers and sources as closely synchronized as possible. The sources could periodically beacon their current time. However, we want to limit the communication to the routers, so we propose implicit synchronization. The source sends out a timestamp of when the seed was generated. After $n \cdot T$ seconds from the timestamp, everyone moves to the next key. Here n is the current position in the reverse hash chain and T is the key validity duration. Clock synchronization is achieved through any of the standard time protocols, such as NTP. While not providing 100% accuracy, the source and routers are likely to stay synced on a millisecond granularity.

How much additional storage is required by routers? When validating the HMAC, the routers look up the prefix and get the key. However, because each key is unique at the AS level, we must store at most 2^{16} keys. The routers also need to look the key up quickly. Routing tables currently store routes based on prefixes advertised by ASes. Since we are protecting keys based on prefixes registered by ASes, our total prefix size should be no greater than the forwarding table size. Therefore, we can use the same lookup method (a trie system) and space requirements as for the forwarding table.

We protect both initial and subsequent seed messages using public key encryption, with public keys distributed out of band. The expenses of public key cryptography and storing and updating the reverse hash chain can be paid by the route processors used to handle routing updates, with results sent to routers via out-of-band control paths.

IV. EVALUATION

We evaluated RAD with a series of micro-benchmarks to determine RAD's overhead, and with core traces to replay traffic during an attack, measuring the percentage of successful transactions. We used the DETER testbed [34] and core traces from MAWI [35].

A. Overhead Tests

These experiments focus on L-RAD, as the overhead considerations of C-RAD have been discussed in Section III. We must determine the overhead incurred by TCP from L-RAD on the source machine. If it is too costly to generate the MACs for SYNs or to validate them, we may not gain anything. We need to know how long it takes to calculate our SHA-1 MAC, how long to generate and send a SYN, how long to process a SYN/ACK, and how long to send a RST. Using a Dual Xeon 3 GHz machine running an instrumented Linux 2.6.12-1 kernel, we generated 20,000 SYNs, responded to 20,000 SYN/ACKs and generated 20,000 RSTs. To time calculation of a SHA-1 MAC, we ran one million iterations and calculated the average run time. The results are shown in Table I.

It takes, on average, 16.15 μs to send a SYN, 9.23 μs to process a SYN/ACK and 1.44 μs to run SHA-1. Therefore, we are adding an 8.92% overhead to the SYN generation, and a 23.39% overhead to the SYN/ACK processing. In our traces, only 4.14% of our TCP packets are SYNs, in line with Shin et al. [27], who found that 3.5% to 4.5% of packets are SYNs. 2.10% of our packets were SYN/ACKs. Thus, on average, we are increasing the overhead of TCP by 0.86%. Furthermore, since we drop invalid SYN/ACKs and do not generate RST packets, we do not pay the RST cost. As machines improve in speed and specialized cryptographic hardware becomes more prevalent, the cost of running RAD should also decrease.

What about the source AS' costs for C-RAD? Fortunately, end host ASes are not nearly as speed-dependent as core routers. These ASes can do more processing, as shown by recent traffic shaping endeavors [8]. Speeds of 1 Gb/s should be more than fast enough for end host ASes. Finally, the end host ASes need not keep much state. They only need the current key, their position in their hash chain, and the timestamp.

B. Replay Experimental Setup

To determine the effectiveness of both RAD schemes, we tested them with real traffic, with and without attacks. We replayed backbone traces from MAWI on the DETER testbed. We processed the traffic with a specialized parser using CAIDA's CoralReef package [33]. The parser searches the trace for two-way TCP flows, generating session info for each flow. The IP addresses for all end hosts in the flows are then grouped into sets whose members do not communicate

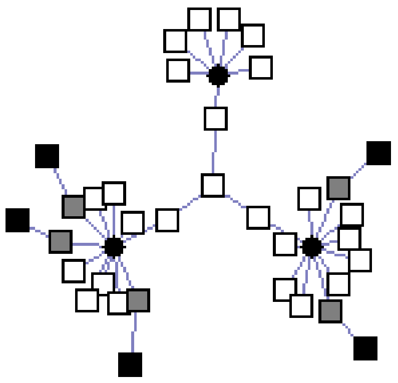


Fig. 2. The Replay Setup. In this diagram, black nodes represent the target nodes and gray nodes represent the gateway nodes. The white nodes are the remaining nodes, which participate in the traffic replay and generate attack traffic, but are not being measured.

with each other. We used SEER [26] to map each set of IP addresses to a single node on DETER; thus, the number of sets we generated was based on the number of nodes used in the replay.

We used SEER's replay tool to replay traffic. This tool is similar to Swing [30]. Like Swing, it replays traffic in a congestion-responsive manner with each connection state dictated by the TCP implementations on the end machines, but with some differences : (1) Swing uses derived distributions of traffic parameters to create new traffic during replay. SEER's replay tool extracts connection dynamics features from backbone traces and replays connections exactly as they occurred in the trace. (2) Swing has a stricter criteria for selection of connections to replay. (3) Swing uses the 10.0.0.0/8 address range, while SEER keeps original IP addresses and service ports from the trace.

As Figure 2 shows, the nodes are clustered into three groups. Each node is connected to a router, which connects to a central monitoring and measurement machine. Attack nodes and reflector nodes have been added. The attack nodes generate spoofed traffic and send them to the reflectors. The reflectors respond to the requests and send replies to the spoofed victim machine(s).

Each trace we use contains fifteen minutes of traffic, an average of roughly 6.4 million TCP packets. We use ten different traces for a total of 150 minutes worth of replayed traffic. The traces are split into twenty nodes, giving each node an average one-way throughput of roughly 500 kb/s. Five of the twenty nodes are randomly selected to be measured during different attack levels, giving a total of 750 minutes worth of traffic data to be measured. We only use five nodes because we are trying to determine the benefit of RAD to a single machine. If we attack and measure too many nodes, the cumulative attack will affect the overall traffic and the results will not accurately reflect the situation we are trying to measure.

The five nodes are given gateway routers that are used in our L-RAD gateway filtering experiments and our C-RAD experiments. Each gateway runs the Click Modular Router [13], with filtering performed by our Click element. The gateways have twice the bandwidth of the nodes, to simulate the situation where a target has multiple gateways or a gateway with more bandwidth. All other nodes besides the gateways and the targets have gigabit connections, to ensure that congestion at these nodes does not affect the outcome of our measurements.

We use Tcpdump [36] to collect the packets at each target node, parsing the dump to determine the number of successful transactions. A transaction is a payload packet and a corresponding acknowledgement. A successful transaction is any transaction that avoids congestion control (i.e., three duplicate acknowledgements or a timeout retransmission). We filter out the failed transactions, as well as the excess caused by the failed transaction (e.g., duplicate ACKs), and count the successful transactions.

C. Local RAD Experiments and Results

For Local RAD replay experiments, each of the five target nodes is given 5 Mb/s of bandwidth, about ten times the bandwidth that their average throughput requires. The gateways have twice the bandwidth, or 10 Mb/s. In this group, three experiment sets are run: one without RAD, one with L-RAD local filtering, and one with L-RAD gateway filtering. For each set, we run five attack patterns per target: no attack, 3,000 packets per second (pps), 6,000 pps, 9,000 pps, and 12,000 pps. We aggregate our data for each attack pattern by summing the total successful transactions for each node, then summing all the successful transactions across traces, giving us the total successful transactions by all target nodes from all traces for each attack. We normalize these values by the total successful transactions from the no-attack data set, giving us the percentage of successful transactions for each defense. We tested two types of attack traffic, TCP SYN/ACK reflection and a DNS response reflection. Figure 3 shows the results from the TCP SYN/ACK test.

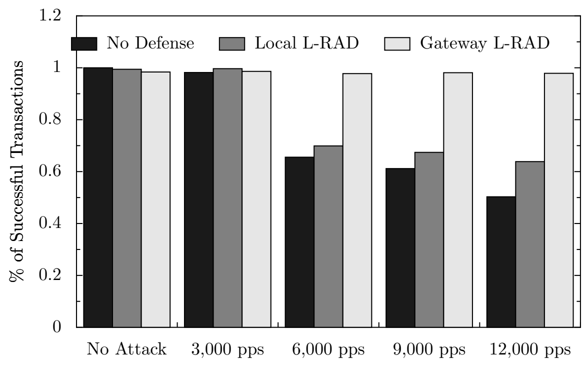


Fig. 3. Percentage of Successful Transactions During a TCP SYN/ACK Flood Using the Local RAD Defense.

When not under attack, all three schemes work well. Local filtering has a 1% performance loss due to the computation cost. In the 3,000 pps attack, none of the schemes suffer, because the attack is not large enough to affect TCP. As the attacks increase, TCP congestion control kicks in and the number of successful transactions begins to decrease. Local filtering performs well, providing about a 15% gain in the largest attack, because RAD drops traffic before having to process it for TCP state, and because it does not send out RST replies. Suppressing RSTs may amplify the inbound attack by causing SYN/ACKs to get retransmitted, but it quashes a locally generated outbound flood of RSTs. Recall that a RST is sent for every unexpected SYN/ACK, so during a 12,000 pps SYN/ACK flood, the outbound links are also being flooded with 12,000 pps of RSTs.

Gateway filtering performs best in nearly all cases, with an approximate average of 98%. Due to Click limitations, the performance cannot reach 100%. Since the gateway has more bandwidth, it can absorb a larger attack. In the last attack (12,000 pps), the attack is so large that it begins to overwhelm the gateway, but at a significantly lower level than when the target deploys the defense. In our tests, gateway filtering outperforms no defense by 30 – 50% and outperforms local filtering by 25 – 35%, depending on the attack size.

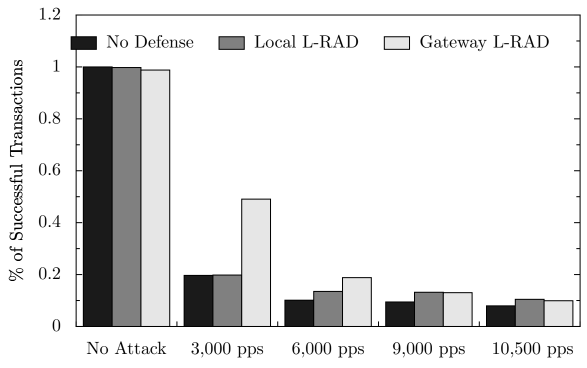


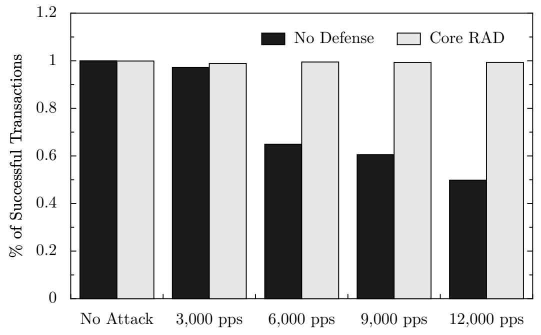
Fig. 4. Percentage of Successful Transactions During a DNS Response Flood Using the Local RAD Defense.

We also tested a reflected flood using DNS, demonstrating reflector attack amplification. In our experiments, we send out a 58 byte DNS query for the address “www.yahoo.com.” The DNS response for this query is approximately 400 bytes. This amplifies our DNS query by about seven, or 10 compared to a non-amplifying SYN/ACK reflector attack. Our 12,000 pps SYN/ACK flood resulted in roughly 5 Mb/s at the target. A 12,000 pps DNS response flood results in about 50 Mb/s at the target. A minor caveat is while we send out 12,000 pps requests in our largest attack, our DNS servers could not keep up, converging at a rate of about 10,500 pps. We again perform all the experiment sets for each trace. The aggregate results are shown in Figure 4.

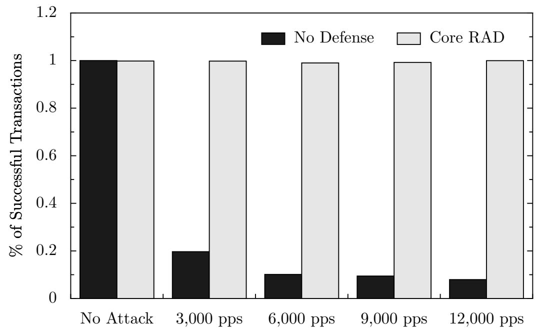
Any attack quickly overwhelms no defense, and local filtering does little better. Once the attack grows past 3,000 pps, it overwhelms the gateway's connection. Local filtering outperforms gateway filtering in the final test due to limitations in Click. If both could operate at the same efficiency, their results should be the same, as in the 9,000 pps attack. Even though L-RAD does not perform well, it outperforms no defense. L-RAD can provide a decent defense against some attacks, but like any “edge” solution, its bandwidth can be overwhelmed.

D. Core RAD Experiments and Results

We made minor changes for the C-RAD experiments. First, each attack packet has a randomly chosen IP ID field. So, on average, 1 in ∼64,000 packets will get past the filters. Second, unlike L-RAD, the gateways in the C-RAD tests generate the HMACs and mark the packets, simulating the case where the target node is an AS, and the gateway is that AS' border router. In this case, the gateways also must send the synchronization messages to the core routers (see Section III). Finally, while there are only 20 nodes in our replay, each node replays hundreds or thousands of IP addresses. Therefore, each core router is responsible for maintaining C-RAD state information for all of these IP addresses, not just 20 nodes.



(a) Percentage of Successful Transactions During a TCP SYN/ACK Flood.



(b) Percentage of Successful Transactions During a DNS Response Flood.

Fig. 5. Percentage of Successful Transactions Using the Core RAD Defense.

Figure 5(a) shows the results of the C-RAD TCP SYN/ACK flood test. C-RAD defeats the attack. There is a 1% drop in performance compared to no defense, which is due to the queuing required to use our Click module. This drop is constant regardless of the attack size.

Figure 5(b) shows that C-RAD defeats the DNS attack, while no defense is overwhelmed. From C-RAD's perspective, the SYN/ACK and the DNS response flood are the same. All it sees is a series of small invalid request packets that it drops.

Our experiments show that our routers filtered 99.98% of the attack request packets. We expected 1 in 64,000 packets, or 0.001% of the packets, to make it through.

V. Deployment

What is an adequate deployment for each system to be effective? Local RAD does not require any infrastructure deployment. Anyone can deploy L-RAD on their system or network to get the protections that it provides. Unlike some other spoofing-related schemes, such as ingress filtering, the benefit of deploying L-RAD is primarily to the sites choosing to deploy it.

The question is more complicated for the Core RAD system. [19] shows that 95% of all Internet traffic passes through just 50 ASes. We could get effective coverage by only deploying C-RAD filters at these 50 ASes. This deployment level is also easier for the source ASes, which only need to coordinate with a small set of core ASes, rather than every core router. Furthermore, if the source AS deploys L-RAD as well as C-RAD, they can provide a defense for the 5% of traffic that does not pass through these 50 ASes.

While further deployment would provide better coverage, possibly converging near 100%, it may not be worth the additional required nodes. The more routers added to the filtering scheme, the more coordination required with the source ASes. Furthermore, it increases vulnerability by allowing attackers to test with closer nodes and therefore smaller RTTs. It also increases the number of nodes that could be compromised, exposing the source's secret.

The ASes required for C-RAD may choose to deploy it to reduce spoofed traffic on the Internet (since spoofed traffic is rarely of value to them or anyone else), or they may choose to offer C-RAD as a service that edge ASes can purchase to obtain greater protection against spoofing-based denial-of-service attacks. Since there exists only a small set of ASes that can effectively filter, they have financial power over the service. Further, while service providers may purchase the service from those Core ASes, they can in turn charge their customers who wish to be protected by the service.

VI. Limitations and Future Work

There are a few limitations to our work. Our experiments only test reflector attacks on TCP traffic, because TCP makes up 80% to 90% of our test traffic. To test the effect on DNS, for example, we would have to artificially generate legitimate DNS requests, which may not represent reality. Also, the process we use to divide the trace into nodes and replay traffic only currently works for TCP traffic. While our scheme should work with the majority of traffic types, we must evaluate it on alternative legitimate traffic types.

L-RAD violates the TCP specification [25]. According to the standard, TCP should send a RST in response to an unexpected SYN/ACK. We quash this as part of our defense. This action can be justified because the SYN/ACK is perceived as a apparent attack vector and we are simply defending against it. We could reduce the problem by having RSTs sent until the unexpected SYN/ACK volume gets too high, after which they are quashed, handling the case where an unexpected SYN/ACK is received rather than a reflector attack. We could also modify the gateways to send RSTs in a similar manner.

Another limitation of L-RAD is the occasional legitimate retransmission of a SYN/ACK that misses the counter window. The MAC contained in this retransmitted SYN/ACK is no longer valid, so it will be lost. We can fix this problem by picking a new initial sequence number (a valid MAC) for the eventually retransmitted SYN. We did not study the effect of lost legitimate SYN/ACKs in our experiments. While it probably occurred during our attacks, it did not noticeably affect the benefits of L-RAD.

C-RAD will not work with IP fragmentation. Because C-RAD uses the IP ID field for its HMAC, any actual fragmentation will invalidate the HMAC. We justify our technique for the following two reasons: First, IP fragmentation is not very common. In our traces, only 0.05% of the traffic was fragmented. Second, IP fragmentation can be prevented with path MTU discovery. Since anyone who is running C-RAD knows this is a limitation, they should also deploy path MTU discovery to overcome this minor problem.

VII. Related Work

Much research has focused on defeating IP spoofing and DoS attacks. Since reflector attacks require IP spoofing, work to defeat IP spoofing defeats reflector attacks. Early work was targeted toward identifying sources. Ingress filtering [9], for example, prevents hosts from spoofing addresses that cannot exist on that subnet. Park and Lee propose RBF [21], leveraging the fact that most of the traffic a router sees from a source comes in on the same interface. If RBF sees packets from that source on a different interface, it filters the packet. Park and Lee analyze the effectiveness of RBF but do not propose a deployment strategy. Duan and Yuan propose IDPF [7] which uses BGP information to build a relaxed RBF table. However, IDPF is slow to respond to routing changes, and may drop legitimate traffic. While RBF allows same path spoofing (i.e., if A and B come from the same interface, A can spoof B), IDPF allows spoofing from multiple possible paths, lowering effectiveness.

Jin et al. propose HCF [12], which works by associating sources with hop counts instead of interfaces. If a packet comes from a source with a different hop count, the packet is dropped. Like RBF, two nodes with the same hop count can spoof each other. Further, if the attacker can guess the right hop count for a spoofed source, he can fool this system. Similar to L-RAD, HCF can only provide limited protection at end hosts before the host is overwhelmed. This scheme could be used in the core; however the core experiences asymmetric routing, which causes this scheme to fail.

SPM [5] and SAVE [15] work by coordinating between senders and receivers. SPM has ASes exchange cryptographic secrets, and then marks packets between them. If an invalid mark is detected, the packet is dropped. The primary problem with this is that only packets between SPM participants can be defended. SAVE works by sending periodic advertisements to all destinations about their prefixes. Filters use the advertisements to learn the arrival interface for that prefix, and then RBF can be used. However, incomplete tables can form via routing changes; these tables can limit the effectiveness of SAVE. Further, SAVE requires complete, or at least contiguous, deployment.

Many defenses for DoS have been used. L-RAD uses similar techniques to SYN cookies [4], which defend against SYN floods. SYN cookies work well against a SYN flood until the target's bottleneck link is overwhelmed by the attack volume, but the technique only protects against this particular attack.

Peng et al. propose a distributed reflector attack defense in [23]. Reflectors monitor traffic for RST traffic. The reflectors raise detection signals when RST traffic becomes too high. The reflector attack mentioned in the introduction can overcome this approach with effective reflector scattering. This approach also cannot distinguish between legitimate and spoofed traffic.

SNF [2] uses a special sequence number similar to the request mark. Any SYN/ACK that does not have the pattern is filtered. The authors propose ISP filtering, where the secret pattern is shared with the ISP, similar to gateway filtering. However, the change interval needs to be reasonably long, on the order of minutes, to reduce the communication overhead, and this increases the vulnerability window. In contrast, L-RAD uses a counter and a secret, where the counter only needs to be time-synchronized after secret dissemination. As with the other security properties discussed in L-RAD, a small counter window leads to a small vulnerability window. Also, the SNF scheme only works with TCP, where RAD works with multiple traffic types. Finally, RAD offers a higher level of security through multiple levels of filtering.

Some methods may require significant changes to infrastructure to defend against DoS. In TVA [31], a source attaches capabilities to each packet, and each router verifies one capability. The capabilities are route-dependent, so if the route changes or multipath forwarding exists, legitimate traffic is dropped. Stack-Pi [24] associates a source address with a mark in the IP header. The mark is jointly created by Stack-Pi routers that forward the packet. During a DoS attack, marks can be used to filter paths with a large traffic volume. The filtering, however, inflicts collateral damage on legitimate traffic on those paths.

BASE [14] leverages capabilities of both Stack-Pi and IDPF, while solving several issues such as route asymmetry. First, they distribute marking values among BASE filters for each source AS. The marking value is computed by hashing a secret key and the mark of the previous filter along the path. When a BASE-enabled AS is attacked, the victim invokes marking and filtering through a BGP update. BASE edge routers create the initial mark for any packet they believe to be legitimate. BASE-enabled transit ASes validate a mark, and then the router replaces the mark with its mark and forwards it. Thus every packet with the same source address will have the same mark when it leaves a BASE node. Since BASE uses BGP updates to invoke filtering, the system is vulnerable to policies that do not forward these updates, leading to improper filtering and dropped legitimate traffic. BASE is also vulnerable to the probing scheme in Section III. Because they only redistribute marks during routing changes, a learned mark can be used until the next routing change on that path. This learned mark can be used for many different packets, unlike C-RAD.

Shue et al. propose a system similar to BASE using tags [28]. Here, routers close to end hosts tag legitimate outbound traffic. When a packet arrives at another filtering node, it validates the current tag. If the tag is valid, it replaces the tag with a new tag. Tags, like BASE marks, are distributed via BGP updates. They solve route asymmetry and BGP policy issues by giving routers the capability for learning tags of other nodes. This learning capability requires some packet inspection, which requires extra processing for core routers. They also discuss multiple options for where to place the tag. If the tag is placed as an IP option, the packet risks being placed on the slow path. If the tag is placed in the IP ID field, the authors' system is vulnerable to the probing technique discussed in Section III.

Both BASE and Shue's system require a minimum level of deployment. Their systems require initial filters close to the edges to determine the legitimacy of the traffic generated there and apply the initial mark. If initial filtering occurs too far into the core, it will be difficult for the initial filter to determine a packet's legitimacy. Thus, neither system could adequately filter by only using the 50 core ASes required by RAD. BASE requires only 30% deployment to obtain reasonable filtering, but this is still on the order of thousands of ASes. While Shue's system requires a smaller deployment then BASE, between 10 – 30%, this is still greater than the 50 ASes required by C-RAD.

Finally, StopIt [17] deploys a new infrastructure service at each AS. When a node is under attack, it sends out a StopIt request. This request is disseminated to routers along the forwarding path and eventually to the source AS. If the source AS is well behaved, it stops the traffic itself. Otherwise, filtering nodes along the path will filter the traffic out. Such a system only works if the deployment level is high and IP spoofing is impossible. StopIt solves the IP spoofing problem using route-dependent passports [18], as discussed in Section III-B. In contrast, neither version of RAD is route-dependent.

VIII. Conclusion

The Reflector Attack Defense defends against reflector attacks both locally and in the core. In Local RAD, message authentication codes mark requests and validate the MACs on the replies. We showed how to generate the MACs using a secure hash, as well as how to validate them. We also showed how sources can synchronize with their local gateways or border routers, so these locations can filter the packets for them. By doing so, the routers allow us to filter the traffic upstream, taking the burden off the targets.

Our L-RAD experiments show that the cost of generating the MAC is acceptable and will decrease as technology improves. Replay traffic experiments show the benefits of L-RAD, both locally and at gateways. Local filtering prevents the use of resources or backscatter associated with an attack. Gateway filtering provides more substantial protection limited only by the speed of receiving and filtering packets.

In C-RAD, we use HMACs to mark all outbound IP packets from an AS. Core routers use the HMAC to validate the source and filter. We show how source ASes and core routers can synchronize the required information to perform filtering. We discuss the speed and scaling issues faced by core routers, and how to overcome them. Our experiments show that C-RAD can completely quash an attack, even when L-RAD is overwhelmed.

The goal of RAD is to allow individual networks more control over their own protection. Unlike ingress filtering, which relies on everyone to do their part, only the local AS and the participating members of the core need to cooperate. We envision dual deployment of Local and Core RAD. Core RAD provides the majority of the protection. Local RAD filters any remaining reflector attack packets that Core RAD cannot filter. Furthermore, Local RAD can protect individual networks or machines without any new infrastructure, even if their ISP or AS is unwilling to deploy Core RAD. By deploying both Core and Local RAD, we can greatly reduce the effectiveness of both reflector attacks and IP spoofing.

REFERENCES

[1] J. Aikat, J. Kaur, F.D. Smith, and K. Jeffay. Variability in TCP Round-trip Times. In *Internet Measurement Conference*, 2003.

[2] D. Basheer and G. Manimaran. Victim-assisted Mitigation Technique for TCP-Based Refelector DDoS Attacks. In *4th IFIP-TC6, Networking*, pages 191–204, 2005.

[3] S. Bellovin. A Technique for Counting NATted Hosts. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, 2002.

[4] D.J. Bernstein. SYN Cookies, 1997.

[5] A. Bremler-Barr and H. Levy. Spoofing Prevention Method. In *INFOCOM*, 2005.

[6] C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In *ASIACRYPT*, 2006.

[7] Z. Duan, X. Yuan, and J. Chandrashekar. Constructing Inter-Domain Packet Filters to Control IP Spoofing Based on BGP Updates. In *INFOCOM*, 2006.

[8] P. Eckerlsey, F. von Lohmann, and S. Schoen. Packet Forgery By ISPs: A Report On The Comcast Affair. *Electronic Frontier Foundation*, November 2007.

[9] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2267.

[10] Y. Hu, M. Jakobsson, and A. Perrig. Efficient Constructions for One-Way Hash Chains. In *Applied Cryptography and Network Security*, 2005.

[11] H. Jiang and C. Dovrolis. Passive Estimation of TCP Round-Trip Times. *ACM SIGCOMM Computer Communications Review*, 32(3):75–88, 2002.

[12] C. Jin, H.Wang, and K.G. Shin. Hop-Count Filtering: An Effective Defense Against Spoofed DDoS Traffic. In *10th ACM conference on Computer and Communications Security*, 2003.

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M.F. Kasshoek. The Click Modular Router. In *ACM Transactions on Computer Systems (TOCS)*, 2000.

[14] H. Lee, M. Kwon, G. Hasker, and A. Perrig. BASE: An Incrementally Deployable Mechanism for Viable IP Spoofing Prevention. In *ASIAN ACM Symposium on Information, Computer and Communications Security*, 2007.

[15] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang. SAVE: Source Address Validity Enforcement Protocol. In *INFOCOM*, 2002.

[16] R. Lien, T. Grembowksi, and K. Gaj. A 1 Gbit/s Partially Unrolled Architecture of Hash Functions SHA-1 and SHA-512. In *RSA Cryptographers' Track*, 2004.

[17] X. Liu, X. Yang, and Y. Lu. To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets. In *ACM SIGCOMM*, 2008.

[18] X. Liu, X. Yang, D. Wetherall, and T. Anderson. Efficient and Secure Source Authentication with Packet Passports. In *SRUTI*, 2006.

[19] J. Mirkovic and E. Kissel. Comparative Evaluation of Spoofing Defenses. Technical Report ISI-TR-655, Information Sciences Institute, 2009.

[20] D. Moore, C. Shannon, D.J. Brown, G.M. Voelker, and S. Savage. Inferring Internet Denial-of-Service Activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2), May 2006.

[21] K. Park and H. Lee. On the Effectiveness of Route-Based Filtering for Distributed DoS Attack Prevention in Power-Law Internets. In *ACM SIGCOMM*, 2001.

[22] V. Paxson. An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks. In *ACM SIGCOMM Computer Communications Review*, 2001.

[23] T. Peng, C Leckie, and R. Kotagiri. Detecting Refletor Attacks by Sharing Beliefs. In *IEEE Global Communications Conference*, 2003.

[24] A. Perrig, D. Song, and A. Yaar. StackPi: New Packet Marking and Filtering Mechanisms for DDoS and IP Spoofing Defense. *IEEE Journal on Selected Areas in Communications*, 24(10):1853–1863, October 2006.

[25] J. Postel. Transmission Control Protocol. RFC 793.

[26] S. Schwab, B. Wilson, C. Ko, and A. Hussain. Seer: A Security Experimentation EnviRonment for DETER. In *DETER Community Workshop on Cyber Security Experimentation and Test*, 2007.

[27] S. Shin, K. Kim, and J. Jang. Analysis of TCP SYN Traffic: An Empirical Study. In *International Conference on Advanced Communication Technology*, 2005.

[28] C. A. Shue, M. Gupta, and M. P. Davy. Packet Forwarding with Source Verification. *Computer Networks*, 52(8):1567–1582, 2008.

[29] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *ACM SIGCOMM*, 2002.

[30] K. Vishwanath and A. Vahdat. Realistic and Responsive Network Traffic Generation. In *ACM SIGCOMM: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, 2006.

[31] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *ACM SIGCOMM*, 2005.

[32] Advanced Network Architecture Group. ANA Spoofer Project. http://spoofer.csail.mit.edu.

[33] CAIDA CoralReef. http://www.caida.org/tools/measurement/.

[34] DETER testbed. http://www.isi.edu/deter.

[35] MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp.

[36] Tcpdump. http://www.tcpdump.org/.