Tracefilter: A Tool for Locating Network Source Address Validation Filters[*]

USENIX Security '07 Poster

Robert Beverly
MIT CSAIL
rbeverly@mit.edu

Steven Bauer
MIT CSAIL
bauer@mit.edu

1. BACKGROUND

The Internet architecture includes no explicit notion of authenticity and forwards packets with forged headers. Malicious users capitalize on the ability to "spoof" source IP addresses for anonymity, indirection and amplification [11].

As good Internet citizens, many networks implement source address validation best common practices [6, 1]. However, current anti-spoofing filtering techniques are hindered by an incentive problem. A provider can follow all best practices and still receive anonymous, malicious traffic from third-parties who do not properly filter. As a result, both previous research [2] and recent attacks, e.g. [7, 10], demonstrate that source address spoofing is a viable attack vector.

Filtering presents a conundrum for network operators. Conventional wisdom dictates that ingress filtering is performed near the edges of the network rather than the core. Within the core, filter lists become unmanageably large, or may block legitimate traffic due to multi-homing and routing asymmetry[1]. However the edge of the network contains the largest number of devices and interfaces. Thus appropriately deploying, managing and maintaining these filters is operationally challenging. For example, Bush et al. show more than 10% of all autonomous systems filter traffic to and from newly assigned address space [5].

This research examines *where* in the Internet source address validation filters are employed. We introduce "*tracefilter*," a new, novel filter location technique[2]. Our initial results from live Internet measurements find 80% of filters within two hops of sources.

2. TRACEFILTER

The time-to-live (TTL) field in an IP packet is decremented by each router along the forwarding path in order to prevent routing loops. When the TTL reaches zero, the packet expires and the router generates an ICMP TTL exceeded message back to the source of the packet [4]. In the same spirit as traceroute [8], tracefilter depends on TTL expiration and ICMP.

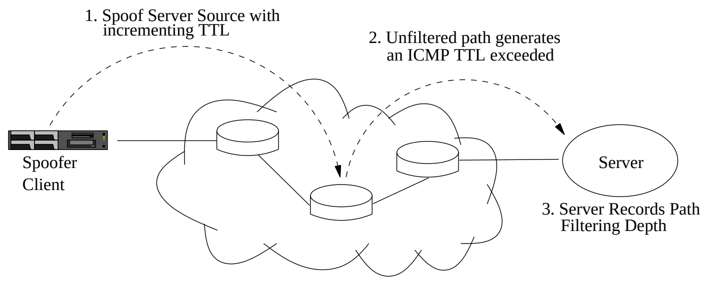
1. Spoof Server Source with incrementing TTL
2. Unfiltered path generates an ICMP TTL exceeded
Spoofer Client
Server
3. Server Records Path Filtering Depth

Figure 1: Tracefilter localizes source validation filters: 1) Client sends spoofed packet with TTL=2, src=S, dst=$S+1$. 2) With no filtering along first two hops, the packet expires, generating an ICMP TTL exceeded message to the server. 3) For each originating TTL, the server records which spoofed packets are received. Client tests entire path; the largest TTL indicates filtering point.

Tracefilter works in conjunction with a server S we maintain. An invocation of tracefilter on a client C sends spoofed source UDP packets with TTLs from $1 \leq ttl \leq d$. The spoofed packet's IP source is the server S. In this way, our server receives and processes any ICMP messages the packet generates [3]. The destination address on packets sent by tracefilter is $S+1$, an IP address on the same subnetwork as the server. While the destination address need only be a valid IP address, we use $S+1$ to test a valid, congruent path.

As spoofed source packets are sent into the network, those that are not blocked by a filter elicit ICMP TTL exceeded messages sent to S. A tracefilter run in progress is shown in Figure 1. Tracefilter is testing the second

[*]This work supported in part by Cisco Systems
[1]Less restrictive forms of reverse path filtering allow partial filtering in some cases of asymmetry
[2]We thank John Curran for the fruitful conversation which germinated the idea for tracefilter
[3]While tracefilter is a measurement utility, a malicious party could spoof in this way as an ICMP DoS technique.

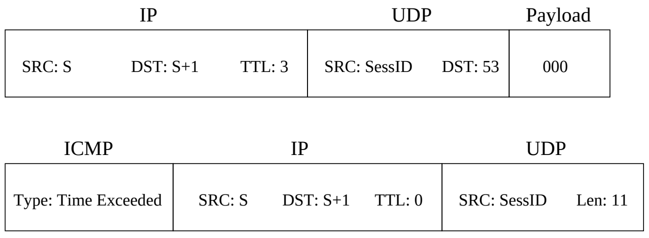

Figure 2: Tracefilter packet format and resulting ICMP TTL exceeded messages. To encode an originating TTL of 3, the tracefilter UDP packet contains a three byte payload. To decode the originating TTL from the ICMP message, S extracts the UDP length from the ICMP quotation and computes $TTL = Len - 8$.

hop along the path from C to S for filtering, so sends spoofed source packets with $ttl = 2$. A source address validation filter will determine that S belongs to a different portion of the network, therefore any packets with source S should not have originated from the network to which C is attached and hence should be dropped. If no filtering is in place for $ttl = 2$, the second hop router will generate an ICMP message to S. S then infers no blocking on the portion of the path up to that router.

How can S reliably determine the originating TTL of packets C sends? ICMP TTL exceeded messages include only the first 28 bytes of the original packet (the IP header plus the first 64 bits of payload) [9]. Thus, the ICMP message quotation includes only the IP and UDP headers of the packet that triggered the message.

The TTL of the packet which generates the ICMP TTL exceeded message is by definition zero. To encode the originating TTL, we pad the payload of the tracefilter packets so that the UDP length field encodes the originating TTL value. Our server then recovers the originating TTL of each tracefilter packet by decoding the UDP length field contained within the ICMP message body. By recording the largest received TTL from a client, the server can infer the number of hops along the path from the client to our server where spoofing filtering is employed. Figure 2 shows the format of the tracefilter packets and the ICMP TTL exceeded messages they generate.

A final detail is how C determines the maximum TTL to test. The client could use a fixed maximum TTL or attempt to infer the distance itself via traceroute, but we want a more principled approach. Instead, tracefilter begins by measuring the IP path length between C and S. Tracefilter sends non-spoofed UDP packets with a TTL set to 64 so that our server can infer the IP hop length of the tested path. S extracts the TTL value from the received UDP packets and computes a distance $d = 64 - TTL_{recv}$. The server then communicates this distance d to the client.

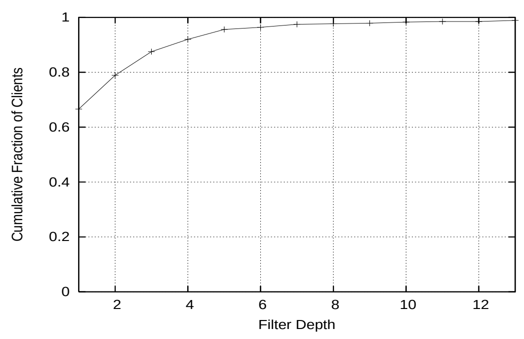

Figure 3: Cumulative distribution of inferred filter depth (IP hops from sender)

3. RESULTS

We implemented tracefilter as part of the ANA Spoofer Project [3]. The results here are from approximately 1300 client runs distributed across the Internet. Figure 3 shows the cumulative distribution of provider filter depth, measured in IP hops from the client, as inferred by tracefilter. 80% of the clients are filtered at either the first or second hop router.

Thus, these results suggest that networks today generally rely upon the edges to properly validate source information. If spoofed packets make it through the first few hops into the network, a spoofed packet is likely to travel unimpeded to the destination.

4. REFERENCES

[1] F. Baker and P. Savola. Ingress Filtering for Multihomed Networks. RFC 3704 (Best Current Practice), Mar. 2004.

[2] R. Beverly and S. Bauer. The spoofer project: Inferring the extent of source address filtering on the internet. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet (SRUTI) Workshop*, pages 53–59, July 2005.

[3] R. Beverly and S. Bauer. The spoofer project, 2006. `http://spoofer.csail.mit.edu/`.

[4] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), Oct. 1989. Updated by RFCs 1349, 4379.

[5] R. Bush, J. Hiebert, O. Maennel, M. Roughan, and S. Uhlig. Diagnosing the location of bogon filters. NANOG 40, June 2007. `http://www.nanog.org/mtg-0706/bush.html`.

[6] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704.

[7] A. Heffernan. Protection of BGP Sessions via the TCP MD5 Signature Option. RFC 2385 (Proposed Standard), Aug. 1998.

[8] V. Jacobsen. Traceroute, 1988. `ftp://ftp.ee.lbl.gov`.

[9] D. Malone and M. Luckie. Analysis of ICMP quotations. In *Proceedings of the 8th Passive and Active Measurement (PAM) Workshop*, Apr. 2007.

[10] NANOG. DOS attack against DNS?, 2006. `http://www.merit.edu/mail.archives/nanog/2006-01/msg00279.html`.

[11] V. Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *Computer Communications Review*, 31(3), July 2001.