Tracing Anonymous Packets to Their Approximate Source

Hal Burch – Carnegie Mellon University
Bill Cheswick – Lumeta Corp.

ABSTRACT

Most denial-of-service attacks are characterized by a flood of packets with random, apparently valid source addresses. These addresses are spoofed, created by a malicious program running on an unknown host, and carried by packets that bear no clues that could be used to determine their originating host. Identifying the source of such an attack requires tracing the packets back to the source hop by hop. Current approaches for tracing these attacks require the tedious continued attention and cooperation of each intermediate Internet Service Provider (ISP). This is not always easy given the world-wide scope of the Internet.

We outline a technique for tracing spoofed packets back to their actual source host without relying on the cooperation of intervening ISPs. First, we map the paths from the victim to all possible networks. Next, we locate sources of network load, usually hosts or networks offering the UDP chargen service [5]. Finally, we work back through the tree, loading lines or router, observing changes in the rate of invading packets. These observations often allow us to eliminate all but a handful of networks that could be the source of the attacking packet stream. Our technique assumes that routes are largely symmetric, can be discovered, are fairly consistent, and the attacking packet stream arrives from a single source network.

We have run some simple and single-blind tests on Lucent's intranet, where our technique usually works, with better chances during busier network time periods; in several tests, we were able to determine the specific network containing the attacker.

An attacker who is aware of our technique can easily thwart it, either by covering his traces on the attacking host, initiating a "whack-a-mole" attack from several sources, or using many sources.

Introduction

One of the major problems on the Internet today is denial of service (DoS) attacks against machines and networks. As opposed to other types of attacks, DoS attacks attempt to limit access to a machine or service instead of subverting the service itself. DoS attacks are simple to design and implement, and there is a plethora of readily available source code which will perform the task. DoS attacks send a stream of packets at a victim that swamps his network or processing capacity, denying access to his regular clients.

There are two basic targets of DoS attacks: machines and networks. SYN attacks [11] are an example of an attack against a machine. In these attacks, a series of TCP SYN packets are sent to a host, filling its table of "half-open" TCP connections. Normal connection attempts are dropped. The basic problem with a skillfully run SYN attack is that the clients and the attackers are indistinguishable without further processing. The server must issue SYN/ACK packets and wait for the client to respond. This particular attack can be mitigated with appropriate algorithms in the server [11]. Other machine attacks may be more difficult to defend against.

The second target type, networks, are much more difficult to defend. Here, the goal is to overload a company's connection to its ISP. The attacker focuses a large stream of data towards the company's network, often from a number of sites. The company's connection becomes congested, resulting in packet loss. Since routers cannot distinguish between attacking packets and valid client packets, they drop them with equal probability. If the attacker can send packets fast enough, the drop rate can become so high that an insufficient number of a client's packets get through. Thus, clients cannot not get reasonable service from any machine beyond the loaded link. The most common of this type of attack is the Smurf attack [8], although recent distributed denial of service attacks (DDoS) [9] have been of this flavor.

The major advantage of DoS attacks is that it is quite difficult to determine the actual source of the attack. Since the attacker can basically put any packet on the local wire, the attacker creates packets whose source IP address is invalid and completely random. Thus, when the victim receives these packets, they are unable to determine the source. The current technique for tracing a packet stream back to the source requires cooperation of all the intervening ISPs. This is something that is difficult to obtain, since the victim is rarely a customer of all of the ISPs between it and the attacker. The standard technique will be discussed in more detail later.

We have developed a method to trace a steady stream of anonymous Internet packets back towards their source. The method does not rely on knowledge or cooperation from intervening ISPs along the path. In additions, tracing an attacking stream requires only a few minutes once the system is set up for a victim.

Basic Technique

We begin by creating a map of the routes from the victim to every network, using any known mapping technology [1, 6, 7]. Then, starting with the closest router, we apply a brief burst of load to each link attached to it, using the UDP chargen service [5]. If the loaded link is a component of the path of the attacking stream, our induced load will perturb the attacking stream. Thus, if the stream is altered when we load a link, this link is probably along the path from the source host of the attack to the victim host. If the intensity of the stream is unperturbed by the load, it is unlikely that the stream of attacking packets is utilizing that link, so we do not need to examine the networks ''behind'' that link.

We continue working back through the network router by router, pruning branches that do not perturb the attack, as we try to narrow the attack source to one network, at which point we can shift to more standard traceback methods by contacting the entity which controls that network.

Executing a trace effectively does require significant preparation in the way of data collection. We need to collect network data, as well as traceroutes from the victim to all possible networks. Due to asymmetric routes, naively, directional data must be collected and maintained by reverse traceroute servers or other means in order to have perfect data. We collect outbound paths and assume that the incoming paths are approximately the reverse of those paths. While this is not completely accurate, by collecting the paths to all networks, we can determine what links could be used on a path from a given network to the victim's network, so this assumption does not cause as many inaccuracy as might otherwise occur.

Because we need to induce isolated load on specific network segments that are not in our purview, we must identify sources ''willing to'' (read: will) perform that task. We recognize that ISPs are now quite regularly turning off the services that we exploit to induce these loads. Thus, we must identify cooperative hosts at the right places in our network map in order to do produce the required load.

This element of the technique is worrisome, since it constitutes a brief denial-of-service attack on that network link. Hackers already employ bulk versions of this approach for denial-of-service attacks. Our technique, on the other hand, carefully limits load to segments only long enough to rule them out as a possible component of the suspected path. The difference is analogous to that between a sword and a scalpel.

In any case, we recognize the antisocial aspect of this technique, and expect that the tool will be used rarely and only in appropriate situations. Possible users include law enforcement, the military, ISPs, and companies policing their own private intranets.

Before attacks or victims are even known, a trusted machine must develop and maintain a current database of networks and load generators. The current version of the tool executes the trace from the victim (targeted) network, but a sufficient complete map of the Internet might allow a neutral third party to run the detecting utility, which would allow flexibility in where to spread some of the bandwidth cost of the tool.

In either case, the tracing machine emits packets that stimulate traffic flow through a desired router or link. A visual display of various statistics of the incoming packets on the victim's network helps determine if that link is used by the packets.

An operator using a tool to probe links on the path back to the attacker. The application of load is done manually (see Figure 1). Though there are algorithms that might automate this process, we require human intervention to reduce the cost of programming errors. We try to supply the operator with information about the amount of load she is inflicting on networks, and she can chose to stop using packet-source networks that have already generated a lot of load.

If the induced load is sufficient to induce drops of incoming packets, it quickly and dramatically affects the attacking flow. The discomfort to ISPs and end users is brief enough that it likely to escape notice. If the load does not induce loss, it may be necessary to run the load generators longer and seek more subtle effects on the workload.

Our technique appears to work better when the network is already heavily loaded, though one can imagine more subtle statistical effects that may be detectable when the Internet is relatively quiet. Our attempts to discover such effects has met with little success. We found we were interacting with cache and other optimizations in various routers. In some cases, our applied load actually increased the packet attack rate!

Assumptions

Our technique does rely on several assumptions, but our experience indicates they are often valid and the technique can work.

Assumptions About the Internet

We assume that most routes over the Internet are symmetric. Asymmetric routes confuse our mapping, traceback and loading. However, the proliferation of reverse traceroute servers, which has proven quite useful for network diagnosis and debugging, might also facilitate construction of at least a partial directional map of routes.

We also assume that we can generate enough load on a particular Internet link to affect performance, in particular loss, statistics of the stream of attacking packets. We must have access to enough packet generators beyond the tested link to load it, which can be challenging across infrastructure with fast links and slower downstream networks. The techniques for doing this will be discussed below.

Hacking Behavior

We assume that the attack is from a single host, at a fairly consistent rate, and runs for a reasonably long time. Denial-of-service attacks are more vexing if they are ongoing, and we have seen attacks that last for weeks. We have seen attack rates of 200-500 packets per second from a single host. We need time to move equipment and programs into place, map routes, and perform the actual traceback.

Bizarre behavior can occur during the traceback, so we have to examine clues carefully. For example, the operator might notice an attacking stream drops by 33% rather than dropping off entirely. Such behavior would be consistent with two or three concurrent attacks from separate hosts; it also possible that the attacking stream is being load-balanced across three different links. Unfortunately, only one packet stream can be traced at a time, so being able to distinguish among the streams would be essential to be able to perform the trace. The operator might be able to use the arriving TTL value, assuming packets within each stream are launched with the same TTL value, and with each stream from different hop distances away. tcpdump's filters provide the tools necessary to isolate such parameters, so that feature of the tool can be used if one of these parameters are sufficient to distinguish between streams.

We assume the attacker does not know that her packets may be traced. An effective hacker attacks from co-opted hosts and never returns to the attacking machine. She hides her trail through a thread of login sessions across many hosts and networks before attacking the target. The denial-of-service attack we target with this tool is a one-way packet flow, which does not rely on interactive login sessions.

We assume that there is something forensically interesting at the source of the attack. The effort of running our tool may not be justified if the result is just disabling one attacking host or convincing one community of computers to enforce ingress filtering

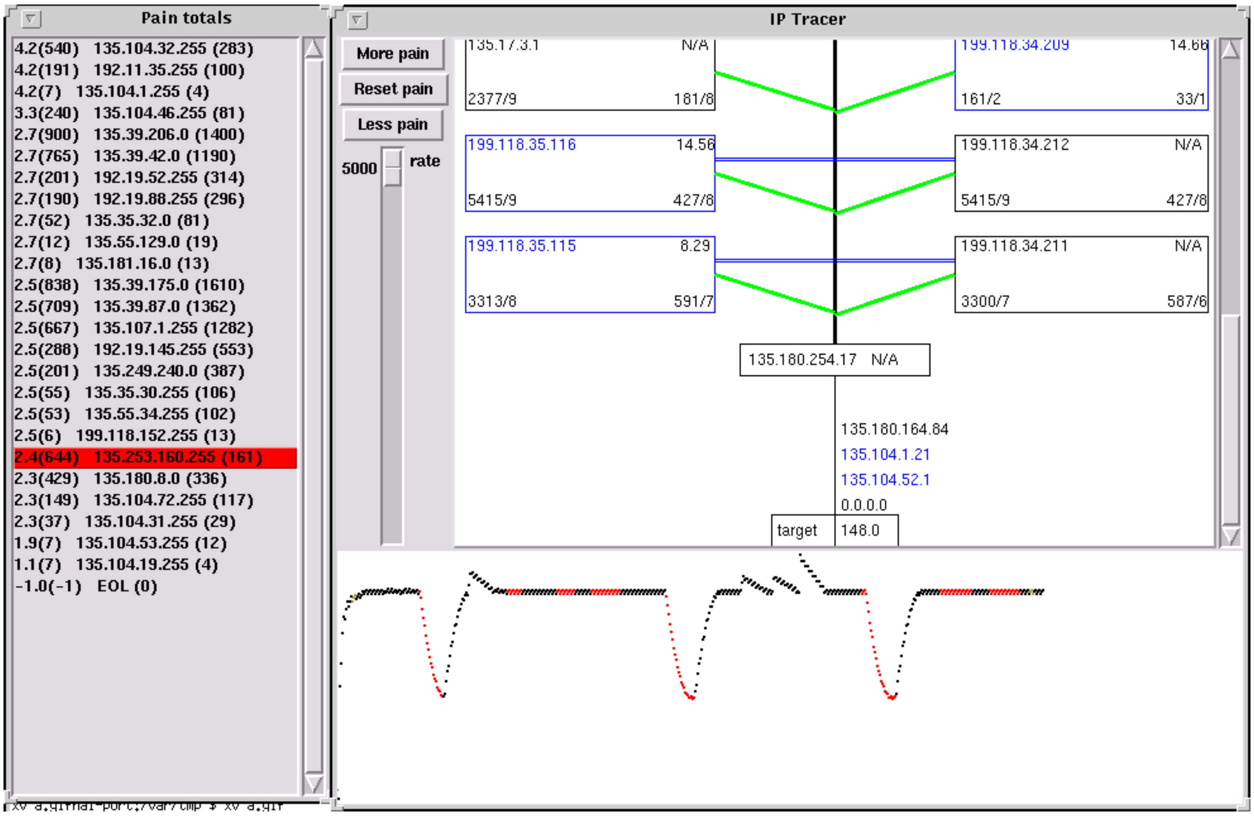

Figure 1: Screen shot of trace-back program. The left-hand screen gives information about the amount the usage of different hosts to generate pain. The bottom is a graph showing the number of packets received per second. The right-top shows the traceback step. The bottom of the traceback shows the path so far, and the top shows the possible next hops. The horizontal double lines are load-balanced lines, so these two IP addresses are really equivalent, for traceback purposes.

[10]. We may be able to catch someone who was not very cautious because he did not expect his packets to be traced. The difficulty of the tracing task renders this a common assumption of hackers.

We also assume that the attacker is unfamiliar with the techniques we provide here. These techniques are easily thwarted in several ways, including modifying the attacking program to vary the source of the attack, altering the frequency of the packets randomly, and attacking from many different sources (the "whack-a-mole" attack).

Network Load: No Gain, No Pain

Once we have determined the path to each network on the Internet, the traceback is done by walking backwards through the resulting directed graph. We load a link and hopefully cause enough packet-loss to see a noticeable drop in the rate of attacking packets. If a significant drop occurs, we can be fairly certain that the tested link is on the path from the attacker to the victim. Otherwise, either the link is not on the path or we did not provide enough load, or 'pain,' to that link to incur packet loss. Note that since most links are full duplex, we need to load the link in the direction towards the victim.

This traceback requires making a high capacity link very busy for a short period of time, on the order of a second. It is difficult to generate a flow of packets from a single host that will do this: it would have to come from a fast host on a fast, unloaded link. We would prefer some leverage, some "gain," on packets we emit. If we send out a flow of x bits per second (bps), we want the resulting flow across the link to be of kx bps, were k is large enough.

To produce the load, we could send a series of messages, such as ICMP echo request (ping) packets [4], from the *victim*'s network out to distant networks whose return path we expect to include the link we wish to load. However, using ICMP echo request packets gets us only one byte in return for every byte we send out, which is a gain of only 1. In addition, the return packets traverse the entire network back to the victim, which loads the entire set of links from the assistant network to the victim, which obscures the data when trying to determine the third link out. Sending ICMP echo requests from a separate network dedicated to this service is also problematic, since the nature of Internet routing means that it is hard to assure that their return path traverses the link we are testing.

Instead of sending packets from the victim's network, we send spoofed packets from a test host located elsewhere on the network. When testing a particular link, we send probe packets to the router on the far end of the link, using as a return address the router on the near end of the link. The near router indignantly discards the unsolicited replies (if using TCP, it actually may reset; for UDP, it may reply with a ICMP Port Unreachable).

More Gain

Many routers make special efforts to put rate limits on handling of ICMP echo requests, since they are used so often. More importantly, the gain of 1 does not help us much anyway. Thus, we need to use a different service in order to supply the load.

The most obvious choice of service to employ is the forgotten tiny service TCP character generator (chargen) [5]. This service generates continuous data to anyone who connects to it, exactly what we want. The rate of data flow is limited in general by the rate that the data is acknowledged by the client machine. At the cost of a few TCP ACKs from our side, we can coax a steady stream of data out of a site supporting this service. Several of these routed over the target link will generate substantial load. We could even use the TCP ACKs to pulse all the transmitters to provide a fine burst of load by ACK-ing several open chargen sockets simultaneously. TCP chargen is turned off on many of the Internet's hosts and routers, but there are many that run the service, and they are easy to fine.

We recognized two major problems: the TCP processing on our local host slows this chargen stream down more than we would like, and, more importantly, the chargen stream still must traverse the path all the way back to the sender, unless we try TCP sequence guessing and IP spoofing, which gets very difficult very quickly. We can circumvent this second problem by using UDP chargen instead of TCP, and spoofing the packets, but this method provide little gain, as we usually get around 102 bytes back for our 40 bytes, a gain of only 2.55. (We include 12 bytes of data in our packets that give information about the actual source of them.) The chargen RFC specifies that the return packet should have between 0 and 512 bytes of data [5] (not counting the 28 bytes for the IP and UDP headers [2] [3]). We found, however, that some Windows NT 4.0 hosts violate this standard and return up to 6,000 bytes in response to a single packet, a gain of 150!

A spoofed ICMP echo request to a broadcast address can yield gain as well. By locating networks 'beyond' the link to send directed broadcast ICMP echo requests to, we get a gain of one for each host on that network which responds. Unfortunately, many routers process broadcast ICMP echo requests in such a way that only the router itself returns a packet. This is, of course, fortunate for the potential victims of broadcast ICMP echo request attacks, and is, in fact, recommended for that reason [8]. However, it limits broadcast ICMP echo request's usefulness to us.

Such routers do let other broadcast traffic through, however, and we found that we could obtain gains in excess of 200 quite often using broadcast UDP chargen packets, even on networks without NT 4.0 hosts. Surprisingly, many networks within Lucent still respond to broadcast address 0 instead of 255, so we had to check both to determine the correct one for each network. Figure 2 shows a distribution of networks and their gain for Lucent's intranet. Note that the networks with a gain of less than 1 have a gain of 0, which means that they did not respond to broadcast UDP chargen at all.

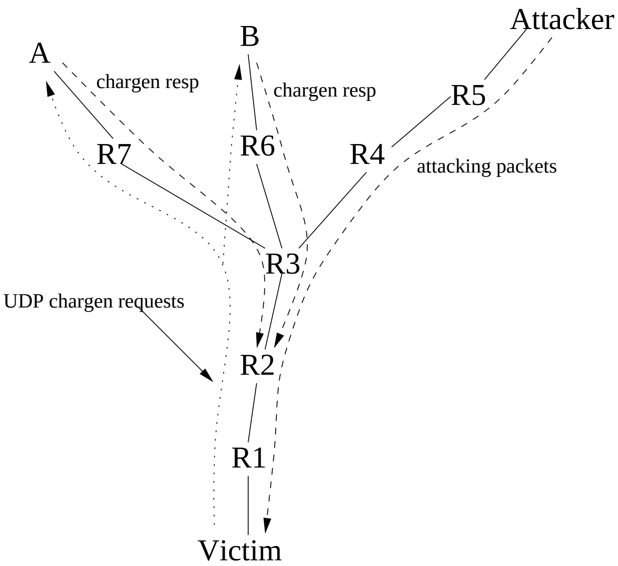

Figure 3: Example of traceback step. Packets are sent to A and B, spoofed from R2, in order to initiate packet flows towards the victim. This causes increases congestion along the R3-R2 link, which, if sufficient, will induce packet loss.

When we initiate the load, the goal is to load one line or, maybe one router. We certainly do not want to load the entire path back to the victim. We prevent this in two different ways. First, as mentioned above, we spoof the return address of the UDP chargen packets to be the address of the router on the victim's side of the link. Second, we utilize multiple UDP chargen hosts. To test a link, we select networks that reside behind the link, as seen from the victim (see Figure 3). In particular, we select networks that have hosts that respond to UDP chargen broadcast packets. We select a network for each outbond link from the far router of the line we are testing. This strategy focuses the load on the line under examination; the packets travel to the machine over different lines, hopefully not affecting each other significantly (again, Internet routing is not inconsistent with their having traversed a common link previously in the path, though it is unusual). The load is limited by the lines the load must traverse, the speed of the networks where the load is being generated, or our ability to emit UDP chargen request packets.

The average gain seen in our experiments is around 133.8 within Lucent. One misconfigured network had a gain of several tens of thousands due to oddities in its configuration (see below). We can easily generate 2,500 40-byte packets per second, or 800 kbps. To flood a 10Mbps Ethernet only requires a gain of 12.5. Figure 4 shows the necessary gains to load a variety of line types. In order to flood a backbone link, such as an OC-48 or OC-192, one needs gains in excess of 3,000, which is larger than all but one of the gains that we have seen. However, when loading backbone links, we have help from the rest of the traffic that is traversing those links, so the actual amount of traffic required to start packet loss is much less than the number in the table. Also, we could increase the

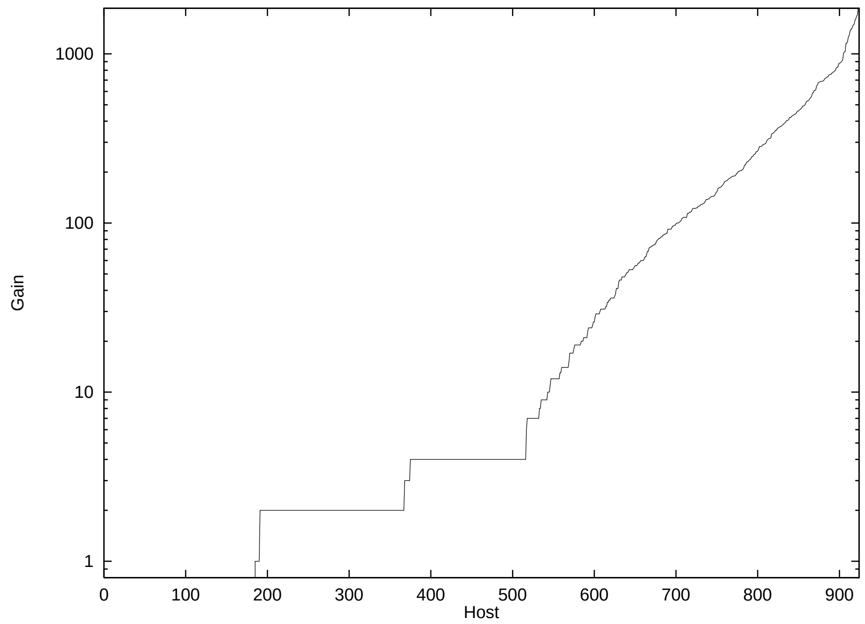

Figure 2: Distribution of gains seen using the broadcast address for Lucent's intranet. The network that generated a gain 43,509 and is excluded from this graph.

rate of outbound packets greatly by using multiple computers that connect to the Internet over different links.

Line Type	Gain Required
10Mbps Ethernet	12.5
100Mbps Ethernet	125
T1	1.9
T3	56
OC-12	777
OC-48	3,110
OC-192	12,441

Figure 4: Required gains to load a variety of line types, assuming 800 kbps of emitted packets.

Note that these numbers are a bit rough, since some of those 2,500 packets will most likely be dropped. Also, we could use 28 byte packets instead of 40 byte ones, but it is not clear that we could transmit them much more quickly.

We have discussed only one possible technique for loading the actual line; another possibility is to load the router. Diverting packet flow by sending a message directly to a router is quite difficult, as Internet backbone routers ignore various ICMP messages to redirect or stifle packet flow. Most methods to load a router have to tackle its system configuration to limit return data flow. Router designs also typically have almost all forwarding handled by a simple machine that just delegates difficult tasks to a higher layer. Less legitimate options, such as hijacking BGP sessions or breaking into the router itself are much too malicious to be seriously considered.

There are other possibilities on ways to slow routers, however. One option is to ping flood the router, i.e., send it ICMP echo requests as fast as possible. A similar alternative is to send the router a flood of packets whose Time to live (TTL) value expires at the desired hop along the path, or to transmit a stream of UDP packets to high ports to stimulate responding UDP port unreachables. Since most routers seem to rate-limit UDP port unreachable messages, we abandoned this idea before testing it extensively. The other methods do not seem to have a major effect.

Another idea is to spew packets at the router to try and upset its routing table. That is, find some sort of packet it responds regularly to (TTL exceeded, echo request) and send it a bunch of packets with random return addresses. Coping with the packets will require enough attention to unsettle the route table cache. In order to combat the incoming stream, it may be useful to pick a handful of sources and cycle through them. This approach has not shown much promise when used within Lucent, perhaps because many Lucent routers use only a single default route so forwarding cache state is not a resource issue.

Results

We obtained logins on various hosts throughout Lucent's intranet. We ran a non-privileged program named `sendudp` to generate a stream of packets back to a nonexistent host our local network. In most cases we could trace the packets back to the "attacking" building. In many we could traceback to the individual Ethernet.

Some links did not respond to our applied load. In some cases we had to go a hop beyond the non-responding links (all the links that are connected to a machine that are one hop away from where we had traced back to) in order to find a link which, when loaded, affected the packet flow. Sometimes, we could pick up enough of a signal from one of these next layer links that we could continue. It was a quite manual process, however, which could become difficult on untestable links with a large number of incident links.

With two exceptions, corporate users appeared to be ignorant of our tests. If the mapping is subtle, and the load applied for short periods, users are unlikely to notice the performance hit, or dismiss it as normal network variability.

Early on, we confined our testing to a few networks, and the network administrator received enough complaints to notice our activities several times. Our subsequent tests appeared to be unnoticed, though in neither case did we attempt to hide our activities.

On one network which we used to generate load, the broadcast UDP chargen packet initiated a broadcast storm on their network. This network had a gain in the tens of thousands. Local users definitely noticed every time we used it, since it brought the network to a halt. Of the 2,000 networks in Lucent, only this one appeared to be unstable in this matter. It is unlikely that our probe packets would be detected on such a poorly-run network, which is likely to have frequent packet storms from other causes. The Internet likely has an even lower rate of misbehaving networks.

Alternative Strategies

This solution is not the only possible one to DoS attacks. Since DoS attacks rely on anonymity, a solution must eliminate some anonymity of hosts. There are two basic methods to do this: ensure that sufficient spoofed packets are never transmitted over the Internet and developing a method for tracing back packets if necessary. The first two methods discussed below attempt to stop some of the spoofing, by ensuring that the at least the source IP address is on the same network as the actual source of the packets. The last three methods discuss alternative methods of tracing packets.

The problem with many of these is that they require universal deployment in order to work. If a couple ISPs opt to not follow the method, then the attacker can just launch the DoS from such a network.

Filter Return Addresses at the Source

There are many ways to solve the problem of anonymous packets. The most desirable is to enforce correct source addresses at or near their source via a method called ingress filtering [10]. A company or university should block outgoing packets that do not have appropriate return addresses. ISPs should have similar filters for each of their customers. Many firewalls do this as a matter of course.

This solution is undoubtedly the right one. Anonymous packets have no place on the Internet. However, these filters do make life more complicated, and for large users behind slow routers they can even degrade performance. For network administrators, these filters are an additional administrative problem: one more thing to install, maintain, and get wrong. Several RFC's recommend it as essential for any responsible participant in the global routing system. Most firewalls have the ability and capacity to perform these checks. The source-based filtering may upset mobile networking methodologies.

Filtering in Backbone Routers

Routers at the core of the Internet, those running BGP4 and exchanging full Internet routing tables, inherently enforce proper destination addresses on packets, since the routing system is built around forwarding the packet toward the value of this field. Theoretically, routers could perform a similar check on the source address, i.e., drop those with source addresses that are inconsistent with their incoming interface.

Unfortunately, the verification is not nearly so simple, since a packet may come from more than one possible incoming interface, so routers would have to maintain a huge amount of state. Not only do routers not have spare memory resources to maintain this state, they do not have spare CPU resources to perform the verification. In the midst of sustained forwarding rates of millions of packets per second, often operating quite near if not at their maximum capacity, router designers must optimize for speed. An additional lookup of the source information would require similar optimization, and subsequent re-engineering of many routers, an expensive and unlikely scenario unless ISPs are willing to pay for it.

One could imagine that legal fallout from a particularly damaging attack might force this scenario, and some routers may emerge that support such functionality service without re-engineering. In general, however, the industry has long resisted source-based policy routing, and we do not expect a fundamental change in this mind-set in the short to medium term.

Tracing by Hand

The obvious ad hoc solution to finding a spoofing host is to trace packets back to their physical source manually. This is done by contacting an ISP and having them test each link to determine if a large number of packets are traversing that link destined for the victim network. This is done is in a tree-like manner similar to ours, or at the access points to their networks. There are two basic methods to do this, either examine the traffic flow across the link, or manually disconnect a link and see if it alters the packet flow (essentially what we attempt to do without physical access).

This method requires significant cooperation and attention from intervening ISPs, which has proven a problem in past incidents. They may not have the policy, inclination, time, expertise, or the instrumentation to help out. Test equipment may not be available for some locations or links within their network. For example, some Cisco routers have been known to crash if `IP DEBUG` is used under sufficiently heavy load.

Further, the traces may be needed off-hours: the Panix attack started a little after five one Friday afternoon. It may be hard to find someone at any hour at the ISP who can handle the technical details. Sometimes attacks are only solved because a victim happens to be well-connected to admin-able friends at ISPs that are willing to help them out.

This cooperation is very helpful, but selective, and slows the process down immensely. A quicker method would be extremely useful.

Shutting Down a Router

One could imagine sending a message to a router requesting that it drop all packets for a particular destination for a second or so. This interruption would be long enough to detect a break in incoming packets, without noticeably affecting service. Implementing this feature would provide an obvious denial-of-service attack of its own. The router could require that requests be strongly authenticated, but there is no infrastructure present for such validation in the current Internet. In self-defense, a router would have to do a similar rate-limiting as it does with UDP responses, rendering the feature useless for a significant attack. Given the ease that an attacker can hide her attack, it probably is not worth deploying such a service.

Marking Packets with IP Addresses

Another alternative is to place the IP address of all the routers that a packet goes through during its flight across the Internet. This has two obvious disadvantages: it requires CPU time of the routers and it increases the size of packets, especially in the case of routing loops. Both of these could be reduced by having it mark only every 1 in *n* packets through a given interface. If *n* is small enough, a long enough attack would give you the complete list of routers along the path, if not their actual order. If *n* is chosen large enough, the additional router time and packet size increase would be negligible. In practice, one might want to randomly vary N to avoid possible problems with routers synchronizing. If *n* is too small, than the attacker can insert packets into the network that can "fool" your system into misdiagnosing the path. In practice, you may want to keep only one address in a packet at a time in order to simplify the header.

Ethics

We acknowledge that our methods to traceback anonymous packets resemble techniques used by hackers. There are several questions to deal with in this area.

1. *Does the tracking attempt cause more damage than the actual packets?* Obviously if the answer is yes, then we should not pursue the technique. We cannot provide a universal answer to this question; it really depends on the situation. If the anonymous packet stream has shut down the daytime service on your web server, it is perhaps not costing you enough to take serious action. If they are crashing your network and denying your customers access to the service you sell them, then perhaps stopping is worth the cost of congesting a few network links for a few seconds (it almost certainly seems so to you). The user of this method will have to make this judgment.
2. *Does leaving a service (such as UDP chargen) enabled on your machine implicitly mean you have given permission to use it?* Our method does not attempt to gain access to private information or crash individual machines, but it does leverage accessible services from private machines. However, these machines have left the UDP chargen enabled (or whatever service is employed).

   The easy answer is yes, but it runs dangerously close to the hacker's defense that running a service with possible security holes indemnifies those who intentionally exploit it. On the other hand, we are not really exploiting a security in an implementation. Indeed, we are following the intended protocol specification exactly. Nonetheless, the essence of our tool is the imposition of a denial-of-service of the attacker's own denial-of-service attack against us.

After much consideration, we must conclude that the appropriate answer to this question comes down to motivation. While a hacker is generally trying to harm the machine, gain access to private information, or journey on an ego trip, our tool is leveraging the machine for a secondary purpose that is helpful to the Internet community.

We recognize that this argument may not be sufficient for some organizations that reside on the Internet.

Discussion

This technique is not ideal, either in efficiency, speed, or impact on other Internet users. We have shown that it works on an intranet, which tends to be a more controlled environment than the Internet itself.

It would be preferable to find a better solution involving ISP coordination and cooperation. Unfortunately, we have to admit that sometimes the perpetrators are *at* ISPs, so an official mechanism that tips them off might be completely impotent. We expect that ISPs will drift toward better solutions as their own clients demand assistance.

Acknowledgments

Alexis Rosen and Simona Nass were very helpful in providing information and access during the Panix attack. Peter Winkler and Diane Litman helped us with statistical analysis of perturbed packets. Tom Limoncelli gave helpful information about Lucent's intranet. Andrew Gross, k claffy, Doug Comer, Mike O'Dell, and Marcus Ranum provided a number of useful insights and comments on the issues and techniques raised here.

Author Information

Hal Burch earned his B.S. in Mathematics, Computer Science, and Physics from University of Missouri-Rolla in 1997 and his M.S. in Computer Science from Carnegie Mellon University in 2000. He is now working on his doctorate at Carnegie Mellon while employed by Lumeta Corporation. He is a coach from the U.S.A. Computing Olympiad. Reach him at hburch@cs.cmu.edu or hburch@lumeta.com; see his web page at http://www.cs.cmu.edu/~hburch.

Cheswick has worked on (and against) operating system security for nearly 30 years. Starting in 1987, he worked at Bell Laboratories on firewalls, PC viruses, network mapping, and Internet security. He co-authored the first full book on firewalls and Internet security with Steve Bellovin. The Internet maps he has created with Hal Burch have appeared on the cover of *Nature*, in *Wired*, and the *National Geographic*. Ches recently left the Labs in a small spinoff, Lumeta Corp., that is mapping and scanning corporate intranets. In his spare time he he launches high-power rockets with his wife, works on exhibits for science museums, and automates his home. Reach him at ches@lumeta.com.

Introduction

[1] Cheswick, B., Burch, H., and Branigan, S., "Mapping and Visualizing the Internet", to appear in Proceedings of USENIX Annual Technical Conference 2000.

[2] Postel, J., "RFC 791: Internet Protocol," The Internet Society, Sept 1981.

[3] Postel, J., "RFC 768: User Datagram Protocol," The Internet Society, Aug 1980.

[4] Postel, J., "RFC 792: Internet Control Message Protocol," The Internet Society, Sept 1981.

[5] Postel, J., "RFC 864: Character Generator Protocol," The Internet Society, May 1983.

[6] Govindan, R. and Tangmunarunkit, H., "Heuristics for Internet Map Discovery," Technical

Report 99-717, Computer Science Department, University of Southern California.

[7] Claffy, K. "Internet measurement and data analysis: topology, workload, performance and routing statistics," NAE '99 workshop

[8] CERT, "smurf IP Denial-of-Service Attacks," CERT advisory CA-98.01, Jan, 1998.

[9] CERT, "Results of the Distributed-Systems Intruder Tools Workshop", The CERT Coordination Center, Dec, 1999.

[10] Ferguson, P. and Senie, D. "RFC 2267: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," The Internet Society, Jan, 1998.

[11] CERT, "TCP SYN Flooding and IP Spoofing Attacks," CERT Advisory CA-96.21, Sept, 1996.

[12] CERT, "IP Spoofing Attacks and Hijacked Terminal Connections," CERT Advisory CA-95.01, Jan, 1995.