An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks

Vern Paxson

AT&T Center for Internet Research at ICSI
International Computer Science Institute
Berkeley, CA USA
vern@aciri.org

Abstract— **Attackers can render distributed denial-of-service attacks more difficult to defend against by bouncing their flooding traffic off of *reflectors*; that is, by spoofing requests from the victim to a large set of Internet servers that will in turn send their combined replies to the victim. The resulting dilution of locality in the flooding stream complicates the victim's abilities both to isolate the attack traffic in order to block it, and to use traceback techniques for locating the source of streams of packets with spoofed source addresses, such as ITRACE [Be00a], probabilistic packet marking [SWKA00], [SP01], and SPIE [S+01]. We discuss a number of possible defenses against reflector attacks, finding that most prove impractical, and then assess the degree to which different forms of reflector traffic will have characteristic signatures that the victim can use to identify and filter out the attack traffic. Our analysis indicates that three types of reflectors pose particularly significant threats: DNS and Gnutella servers, and TCP-based servers (particularly Web servers) running on TCP implementations that suffer from predictable initial sequence numbers. We argue in conclusion in support of "reverse ITRACE" [Ba00] and for the utility of packet traceback techniques that work even for low volume flows, such as SPIE.**

I. INTRODUCTION

In a *distributed denial-of-service* (DDOS) attack, the attacker compromises a number of *slaves* and installs flooding servers on them, later contacting the set of servers to combine their transmission power in an orchestrated flooding attack. The use of a large number of slaves both augments the power of the attack and complicates defending against it: the dilution of locality in the flooding stream makes it more difficult for the victim to isolate the attack traffic in order to block it, and also undermines the potential effectiveness of traceback techniques for locating the source of streams of packets with spoofed source addresses.

Figure 1 illustrates the nature of the attack: one host, the master, sends control messages to the previously compromised slaves, instructing them to target a given victim. The slaves then generate high volume streams of traffic toward the victim, but with fake or randomized source addresses, so that the victim cannot locate the slaves.

The problem of tracing back such streams of spoofed packets has recently received considerable attention. In Bellovin's proposed ITRACE scheme, routers (or outboard processors), with a very low probability, send ICMP messages to the destinations of packets they have just forwarded [Be00a]. For a high-volume flow, the victim will eventually receive ICMPs from all of the ITRACE routers along the path back to the slave, revealing its location.

Savage and colleagues proposed a different scheme, in which routers with considerably higher probability mark the packets they process with highly compressed information that the victim can decode in order to detect the edges (pairs of packet-marking routers) traversed by the packets, again enabling recovery of the path back to the slave [SWKA00]. This scheme can trace back potentially lower-volume flows than required for traceback using ITRACE; however, the scheme runs into computational difficulties as the number of slaves increases, a problem addressed by Song and Perrig by supplementing the scheme with the use of network topology maps [SP01].

In more recent work, Snoeren and colleagues discuss a Source Path Isolation Engine (SPIE) that records sets of hashes of packets traversing a given router [S+01]. A victim can then locate the path of a given packet by querying routers within a domain for the set of hashes corresponding to the packet, providing they issue the query soon enough after the packet was transmitted that the record of its presence is still available. SPIE has a major advantage over ITRACE and probabilistic packet marking in that it can facilitate traceback of even low volume (including single packet) flows.

The use of hundreds or thousands of slaves can both greatly complicate traceback (due to the difficulty of disentangling partial traceback information relating to different

To appear in *Computer Communication Review* 31(3), July 2001.

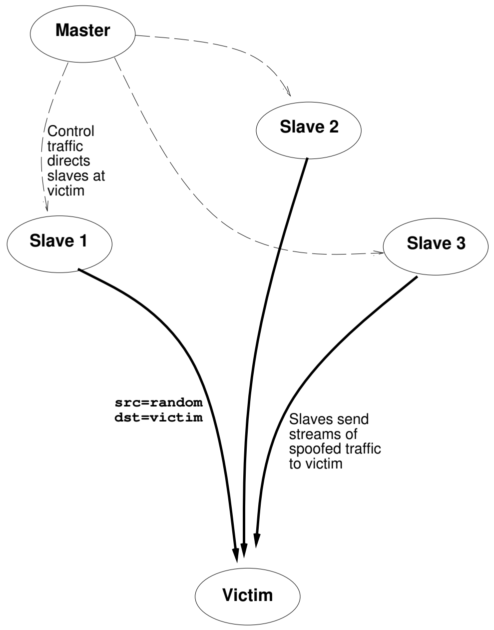

Fig. 1. Structure of a distributed denial-of-service (DDOS) attack.

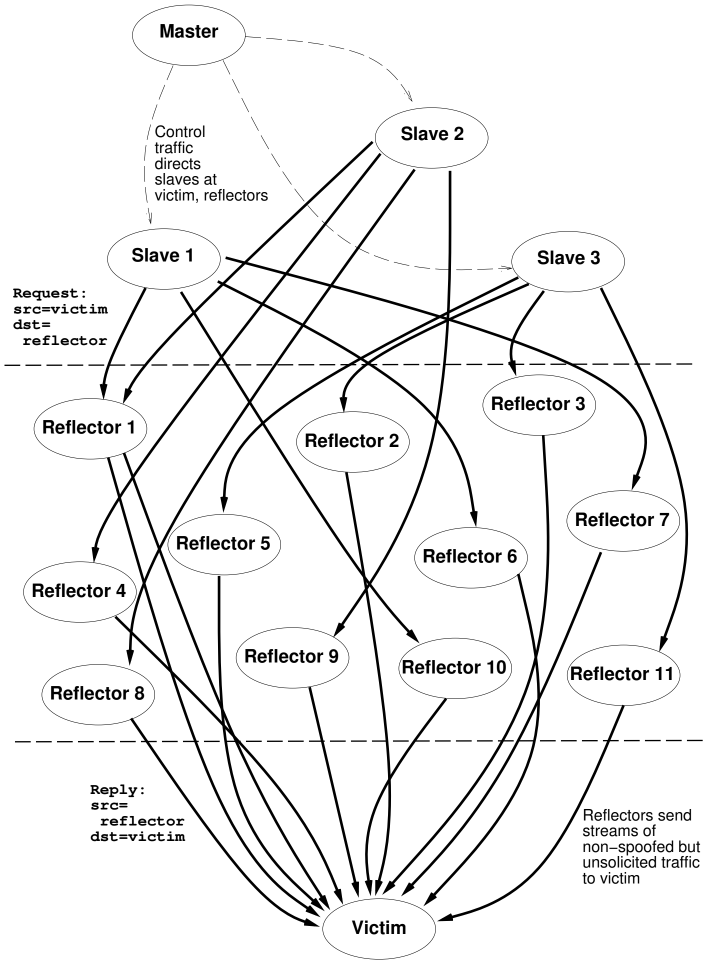

Fig. 2. Using reflectors to render a DDOS attack much more diffuse.

sources, and/or having to contact thousands of routers) and greatly hinder taking action once traceback succeeds (because it requires installing hundreds of filters and/or contacting hundreds of administrators).

Attackers can do considerably better still by structuring their attack traffic to use *reflectors*. A reflector is any IP host that will return a packet if sent a packet. So, for example, all Web servers, DNS servers, and routers are reflectors, since they will return SYN ACKs or RSTs in response to SYN or other TCP packets; as are query replies in response to query requests, and ICMP Time Exceeded or Host Unreachable messages in response to particular IP packets.

The attacker first locates a very large number of reflectors, say on the order of 1 million. (This is probably not too difficult, as there are at least that many Web servers on the Internet; plus, see below on relaxing this requirement.) They then orchestrate their slaves to send to the reflectors spoofed traffic purportedly coming from the victim, V. The reflectors will in turn generate traffic from themselves to V. The net result is that the flood at V arrives not from a few hundred or thousand sources, but from a million sources, an exceedingly diffuse flood likely clogging every single path to V from the rest of the Internet.

Figure 2 illustrates this modification to a conventional DDOS attack. Note that the victim does not require any traceback in order to locate the reflectors; they are readily identified as the source addresses in the flooding packets received by the victim. The operator of a reflector, on the other hand, cannot easily locate the slave that is pumping the reflector, because the traffic sent to the reflector does not have the slave's source address, but rather the source address of the victim.

In principle the operator can use traceback techniques such as those discussed above in order to locate the slaves. However, note that the individual reflectors send at a much lower rate than the slaves would if they were flooding V directly. Each slave can scatter its reflector triggers across all or a large subset of the reflectors, with the result being that if there are N_r reflectors, N_s slaves, and a flooding rate F coming out of each slave, then each reflector generates a flooding rate of $F' = \frac{N_s}{N_r}F$. So a local mechanism that attempts to automatically detect that a site has a flooding source within it could fail if the mechanism is based on traffic volume.

In addition, traceback techniques based on observing large volumes of traffic (ITRACE, probabilistic packet marking; but not SPIE) will fail to locate any particular slave sending to a given reflector. If there are N_r reflectors, then it will take N_r times longer to observe the same amount of traffic at the reflector from a particular slave as it would if the slave sent to the victim directly. Thus, using reflectors provides significant protection against these forms of traceback even if there aren't more reflectors than slaves (or even fewer). Against a low-volume traceback mechanism like SPIE, however, reflectors do not yield such an advantage, and indeed the attacker should instead confine each slave to a small set of reflectors, so that the use of traceback by the operator of a single reflector does not reveal the location of multiple slaves.

Note that unlike some forms of denial-of-service attacks, the reflectors do not need to serve as amplifiers (sending out a larger volume of traffic than the attacker sends to them). They can even somewhat attenuate the volume of traffic sent to them and still serve their purpose effectively. This latitude on not requiring amplification consequently allows a large number of different network mechanisms to serve as reflectors, facilitating the attacker's task of finding a sufficient number of reflectors to launch the attack.

Finally, note that we do not consider here two quite different forms of reflectors: social attacks (in which many people are duped into attempting to connect to the victim) and viral attacks (in which the attacker acquires a vast number of slaves using a virulent virus, and then instructs the slaves to directly attack the victim, perhaps with perfectly legitimate requests [Me00]—the use of reflectors is basically irrelevant, because the attacker already has such an immense number of slaves).

II. Defenses against reflectors

There are a number of possible defenses against reflector attacks:

1. If it is impossible to spoof source addresses in packets, for example by ubiquitous deployment of *ingress filtering* [FS00], then the threat is significantly diminished. The threat does not entirely go away, though, due to the possible use of application-level reflectors such as recursive DNS queries (Section III-E) or HTTP proxy requests (Section III-G), as discussed below. However, while an attacker can still mount a reflector attack even if the slaves lack the ability to spoof source addresses, the victim will be able to more quickly locate the slaves, because if a reflector server maintains logs of the requests it receives, those logs will pinpoint the slave location(s).

For the remainder of the discussion, we assume that we care about preventing attacks in an Internet for which source integrity is not guaranteed.

2. Traffic generated by reflectors may have sufficient regularity and semantics that it can be filtered out near the victim without the filtering itself constituting a denial-of-service to the victim ("collateral damage"). Here, "filter-

ing" refers to the general notion of packet classification; the filtered traffic could then be rate-limited, delayed, or dropped. We will usually, however, presume that filtering means dropping the traffic, and assess the dangers of collateral damage in that context.
3. Bogus reflector requests used to pump reflectors may have sufficient regularity and semantics to enable sites to deploy filtering to prevent their network elements from serving as reflectors.
4. In principle it could be possible to deploy traceback mechanisms that incorporate the reflector end-host software itself in the traceback scheme, allowing traceback *through* the reflector back to the slave.
5. Traffic patterns resulting from a slave pumping a disparate set of reflectors may be discernible to intrusion detection systems monitoring a site's Internet access link.

Of these, we argue that only (2) is potentially viable. We regard (1) as out of scope for the entire discussion of DDOS attacks that utilize spoofed source addresses. (3) requires widespread deployment of filtering, on a scale nearly comparable with that required for widespread deployment of anti-spoof filtering, and of a more complicated nature. (4) has enormous deployment difficulties, requiring incorporation into a large number of different applications developed and maintained by a large number of different software vendors, and requiring upgrading of a very large number of end systems, many of which lack any direct incentive to do so. In addition, (4) may not help with traceback in practice if the traceback scheme cannot cope with a million separate Internet paths to trace back to a smaller number of sources; neither ITRACE nor probabilistic packet marking appears amenable to doing so. (5) requires widespread deployment of security technology at sites which fail to provide such basic security precautions as anti-spoof mechanisms, not a likely combination.

In Section III we examine the viability of (2) in detail, finding that most, but not all, reflector-generated traffic can be filtered without grievously impairing the functionality of most sites. In Section IV we then look at the implications of reflector attacks for traceback, focussing on a modification to ITRACE proposed by Barros [Ba00] and the use of SPIE [S+01].

III. FILTERING OUT REFLECTOR REPLIES

We now turn to an assessment of the practicality of a victim site being able to attain relief from a DDOS reflector flood by filtering out specific types of traffic. To do so, we need to attempt to catalog the different types of reflectors that an attacker might employ.

We assume that the victim (or an upstream provider assisting the victim) can only afford to deploy stateless filtering, given the potentially immense volume of (bogus) state that a flooding attack generates. In principle this could be relaxed to allow some stateful filtering when either the state is highly aggregatable, such as pertaining to particular pairs of interfaces and address prefixes, or the state is instantiated only by traffic from the victim, and hence will not scale disproportionately (unless the attacker can manipulate the victim into violating this assumption). We also assume that the victim, with the cooperation of their service provider, can have such filters installed sufficiently far away from the victim's links that a DDOS attack that targets the access link bandwidth rather than the victim's servers directly can still be throttled, if enough of it matches a particular small set of filters.

Finally, we assume that success in terms of defending the victim is that the filtering allows a significant proportion of the victim's service to continue; we do not require that the filtering leave the service completely unimpaired.

Table I summarizes the analysis developed in the remainder of this section.

A. *IP packets*

We begin by analyzing how the elements of the IP header [Po81a] in arriving traffic might relate to a reflector attack.

The first fields in the header are the **version** and the **header length** (presence of options). Clearly, the version field provides no traction, as the packet won't even make it to the victim unless it corresponds to a version of IP that the victim cares about, and the field is too narrow to likely provide useful filtering. (We confine our subsequent treatment of IP and ICMP to IPv4.) The presence of options similarly does not provide any traction: options are almost never used due to their performance impact, so they won't show up in either legitimate or reflector traffic; even if the attacker can induce their presence, the victim will almost certainly be able to filter out the traffic without impairing themselves.

The **type of service** field in some scenarios could in the future be quite helpful. If traffic to the victim normally arrives with a particular Diffserv Code Point (DSCP) [NBBB98], then likely a static filter allowing only such traffic would help screen out reflector traffic, though at a cost of screening out legitimate auxiliary traffic to the victim, too. This latter could be quite expensive, if it includes things like replies to the victim's DNS queries, or the victim's outbound Web surfing. (Whether the attacker can manipulate a reflector into having a particular DSCP attached to its traffic will depend on the classifiers that wind up being deployed at different points in the network. If Diffserv traffic in general is premium traffic, then it ap-

Protocol/element	*Filtering notes*
IP version	Insignificant.
IP options	Insignificant.
IP TOS / DSCP	Could aid victim if attack traffic non-premium.
IP length	Insignificant.
IP ID	Insignificant.
IP fragments	No cost to filter out unless victim uses fragment-inducing protocols (NFS, AFS, GRE).
IP TTL	None.
IP protocol	None.
IP checksum	None.
IP source	Only filterable if victim can identify as uninteresting.
IP destination	Only filterable if victim can identify as uninteresting.
ICMP request/reply	Likely not difficult to filter out. Includes *smurf* attacks.
ICMP problem	Likely not difficult to filter out.
TCP source port	If filtered, no general access to remote server of given type.
TCP SYN ACK	If filtered, no general access to remote servers.
TCP RST	If filtered, state will accumulate over time.
TCP guessable seq. no.	**Major threat.**
T/TCP	Would be significant threat but easily filtered due to limited deployment.
UDP	No threat due to no inherent reply mechanism.
UDP length	Insignificant.
UDP checksum	Insignificant.
DNS query/response	Can be filtered by opening up holes to specific remote servers.
Recursive DNS queries	**Major threat to name servers.**
SNMP request/response	Generally can be filtered out with little impact on victim.
HTTP proxy caches	A significant threat, but likely easily traced back to slave.
Gnutella "push"	**Major threat**.
Other TCP applications	Will in general be traceable to slave if application server keeps logs.
Other UDP applications	*Unknown*.
Other overlay networks	*Unknown*.

TABLE I
SUMMARY OF DIFFERENT REFLECTOR THREATS AND THE EFFICACY OF COMBATTING THEM USING FILTERING.

pears plausible that often the attacker will not be able to force the marking, because to do so they will have to dupe a classifier upon which billing for the premium traffic relies. Presumably this will be difficult to do, given the financial motivations to secure use of the premium traffic.)

It is hard to see what filtering benefit can be had from the **length** field, since few forms of reflectors will be limited to sending only particular-length replies, and, even if they did, filtering them out would also filter out legitimate traffic. However, if the traffic to a victim tends to come only in particular sizes, such as small DNS requests, then the victim can possibly filter out any reflector traffic that does not come in that size.

The IP **ID** field is very difficult for the attacker to manipulate, and carries only a smidgen of useful semantic information. Thus, it is very hard to see how it could be usefully filtered on.

It is likewise difficult to see how the attacker can take much advantage of **fragments**. It will only be possible to generate them for reflectors that send large replies using TCP/IP stacks that do not implement PMTU discovery [Mo90], or for reflectors that have paths to the victim that transit GRE tunnels [F+00]. Fragments offer the benefit to the attacker of making it difficult for the victim to filter on TCP or UDP header information, since it will only be present in the initial fragments. (Also, that header might itself be split across multiple fragments; some stacks have been observed to do this as part of their normal operation [Pa99].) But due to the limited use of fragments in the Internet, the victim could likely filter out all fragmented traffic and suffer little degraded service, other than for some implementations of protocols like NFS and AFS that frequently send high volumes of fragmented traffic, or for sites that rely on GRE tunnels.

The **TTL** field is easier for the attacker to manipulate (by choosing the distance from the reflector to the victim, and choosing reflectors according to their particular stacks, and hence their particular initial TTLs). But it is hard to see how the TTL can be used for any useful filtering, unless the only legitimate communication the victim partakes in comes from a small set of remote sites that can be characterized with a narrow TTL range.

The **protocol** field will determine the next layer of filtering, as discussed below for particular protocols. Clearly, protocols unimportant to the victim can be filtered out based on this field, so the attacker will need to select reflectors that have one of the same protocol fields as the victim's desired traffic. But this will usually be very easy to do, because the desired traffic will almost certainly include TCP and UDP.

The **checksum** field should provide no traction. It is expensive for a filter to verify, but also appears impossible for the attacker to usefully manipulate.

The final two fields are the **source** and **destination** addresses. Obviously, the same filtering traction applies to these as does for ordinary DDOS floods: if either can be identified as an uninteresting address, then the victim can filter out the traffic; the attacker attempts to ensure that this is not the case. Also note that with a reflector attack, the source address will always be legitimate (Figure 2), unlike with the usual direct-spoofing attack (Figure 1).

Summary: assuming the attacker picks source and destination addresses of interest to the victim, the only angle the victim might try at the IP level is filtering on the DSCP.

B. ICMP

There are two different ways to elicit ICMP reflector replies: using ICMP protocols designed as request/response (such as ICMP echo), or sending traffic that will generate an ICMP message because of some problem associated with the traffic [Po81b].

In the first category are the ping ICMPs (echo, timestamp, address mask, router solicitation, information request/reply). Of these, only the first is widely used, and presumably the victim can get by with little difficulty if replies to all of these are filtered out. (We note, though, that *smurf* attacks, in which the attacker sends ICMP echo requests to subnet broadcast addresses, are essentially a form of reflector DDOS attack.)

In the second category (unreachable, source quench, redirect, time exceeded, parameter problem), the most significant for the victim will be the unreachables, which include *host unreachable* (useful for tearing down state in some circumstance) and *need fragmentation* (necessary for PMTU discovery), and *time exceeded* (needed to run `traceroute`). It appears plausible that the victim would be willing to forgo these as a means to suppress a flooding attack.

Summary: reflectors generating ICMP messages can likely be filtered out.

C. TCP

Rather than walk the TCP header [Po81c], we consider the types of packets that an attacker can coax from a TCP reflector, as these have a great impact on filtering opportunities.

For most victims, what the attacker really wants is a packet that looks like one sent by a true client of the victim: an initial SYN packet, a packet containing data, an acknowledgment, or a FIN. (An additional type, RST, is discussed shortly.)

However, we first note that any packet from a reflector will have a source port corresponding to the port on which the reflector runs. In particular, for Web servers (the most widely available TCP reflector), this will usually be port 80. Accordingly, the victim can filter out any incoming traffic with a source port of 80 (say), and eliminate any threat from TCP-based reflectors. Naturally, this prevents the victim from access to the same service remotely, which may be a significant difficulty; but perhaps an acceptable one during a time of flooding.

Putting that limitation aside, inspecting the TCP state diagram in [Po81c] shows that there is no way to trigger a reflector into sending an initial SYN packet unless it has an application-level means to do so (e.g., FTP "bounce" attack [Ce97]). Furthermore, since the reflector will not have an existing connection open to the victim, the only packets it can send in response to receiving a packet purportedly from the victim are either a RST or a SYN ACK (though see below). If the victim filters out RST packets, this will over time cause its servers to hold more state than they need to, eventually clogging them with stale connections. (However, depending on the service, these connections may be amenable to manual garbage collecting.) If the victim filters out SYN ACKs, then they lose access to remote services (if there isn't more specific port filtering they can employ, per the above); this may or may not be acceptable, depending on the specifics of the victim's operation.

There is, however, another possibility. If the reflector's stack has guessable TCP sequence numbers [Be96], then the attacker can potentially drive the stack through the entire TCP state machine, tricking it into sending data segments, acknowledgments, etc. This is a disaster for the victim. But it is so even without delving into DDOS including slaves and the like—a recently discovered attack exploits such stacks to realize major amplification by duping Web servers into transmitting large items to the victim, and exploiting "ACK splitting" techniques [SCWA99] to greatly enhance the sending rate [Gu01]. Other applications such as FTP or streaming media servers could likewise be exploited. If done as part of a reflector attack, then the attacker gains both the benefits of amplification *and* a highly diffuse flood at the victim, a lethal combination.

Another way an attacker can trick a remote server into sending data packets towards the victim is by forging a T/TCP connection request [Br94] from the victim. T/TCP was designed to resist inadvertent confusion of old network segments with new ones, via the CC/CCECHO mechanism, but the only requirement placed upon the sequence numbers used by the mechanism is simply that they be monotone-increasing. Consequently, it is simple to pick a sequence number in the initial, spoofed T/TCP SYN packet such that the server's stack will find it acceptable. If the SYN packet also contains an expensive request like "`GET hugeimage.jpg`", then the server will begin transmitting the data to the victim immediately.

Three factors limit the severity of the T/TCP attack. First, the T/TCP server will begin in slow start (the specification suggests an initial sending window of 4 KB [Br94], but this is for the client initiating the connection, not the server replying to it). Unless the server's stack has guessable sequence numbers as discussed above, the attacker can't exploit ACK-splitting techniques to move the server out of slow start.

Second, the packets sent from the server to the victim will have CCECHO options in their TCP headers, which makes them amenable to stateless packet filtering, though the filter is potentially somewhat complicated, because due to the use of other TCP options, the location of the CCECHO option in the header will not always be the same. (In addition, such filtering will prevent the victim from accessing external servers using T/TCP; not a significant limitation, however, as they can disable their own use of T/TCP and then the external servers will not use it in reply.)

Third, T/TCP is not widely deployed, so it will be difficult for an attacker to find a large number of T/TCP reflectors. Furthermore, if such reflectors were used in a high-profile DDOS attack, likely many servers would soon be configured to no longer use T/TCP.

Summary: if a site can endure loss of contact to external servers, and can tolerate failing to tear down legitimate connections that the remote peer has reset, then filtering of SYN ACKs and RSTs will protect against the main form of general TCP-based reflector attacks. One exception is if the attacker can find large numbers of remote servers with guessable initial sequence numbers; in addition, due to the amplification of this form of reflection, such servers constitute a potential DDOS threat by themselves, without the attacker even having to coordinate a collection of slaves.

D. UDP

Like IP, UDP is a generic carrier for higher-level protocols [Po80], and by itself does not constitute a reflector threat because there is no inherent "reply" mechanism built into UDP reception. As with TCP above, the port numbers in the header may provide for filtering when an attack is based on reflecting off of UDP servers running on well-known ports. The length and checksum fields appear to provide the same traction as for IP, i.e., essentially none.

E. DNS

DNS servers offer two possibilities for reflection. The first is a reflector simply sending a DNS reply in response to a spoofed DNS request. This form may be recognized because the reply will arrive at the victim from source port 53. Consequently, the victim can filter it out, but at some cost. First, this will impede the victim's own access to the DNS via external DNS servers. Probably the victim can cope with this by opening up holes in the filtering to provide access to a specific set of remote DNS servers, and reconfiguring their local DNS to send queries to them. Second, some DNS queries are made using a source port of 53 as well as a destination port of 53. If the victim provides DNS service, then any such incoming requests would be filtered out. However, by adding filtering on the QR bit in the DNS header [Mo87], such requests can be properly distinguished from the reflector replies.

The second form of DNS reflection concerns DNS servers that in turn recursively query other servers to resolve a request. If the victim is a name server for a particular zone, then the attacker can issue a stream of queries to a large number of name servers that will in turn cause those name servers to bombard the victim server with recursive queries. The queries needn't even be spoofed, which would enable the attacker to launch them in the presence of anti-spoof filtering, though this would reveal the slaves' locations to any monitoring or logging done at the reflectors. But if the queries are spoofed, then the attacker could even use the victim's address as the purported source, such that when the reflector DNS server supplies a reply of some form, that too goes to the victim, a form of amplification (though one that can be filtered out).

Note that caching at the reflector server does not help to ameliorate the attack; the attacker simply keeps changing the domain name used in the bogus query, forcing the reflector to go to the primary name server each time.

Summary: DNS reflection appears to be a serious threat for denial-of-service attacks on name servers. The full degree of the threat depends on whether enough servers support recursion that the second form of reflection is a true threat. Anecdotally, it appears that the answer is yes: a large number of servers do indeed support recursive queries. The only apparent solution to this threat appears to be to include filtering in name servers so that they will only process recursive queries coming from local addresses, coupled with filtering at the site's border to ensure that incoming packets with local source addresses are dropped.

F. SNMP

Another widely deployed UDP-based request/reply service is SNMP [CFSD90]. Sites that fail to block off-site access to SNMP will often provide a large number of possible reflectors, potentially much greater than the number of Web servers or DNS servers with recursion enabled.

However, this attack will be identifiable because it comes from the well-known SNMP port (161). In addition, it seems quite plausible that most victims can survive just fine if external SNMP traffic is filtered out and fails to reach them. On the other hand, this could potentially be a major problem for service providers who rely on SNMP to manage their network. However, they can likely allow replies from their own hosts to pass through the filter, assuming their hosts are numbered out of only a few network prefixes, and thus are easy to express as filter exceptions.

Another question regarding this attack is how many sites do in fact fail to block incoming SNMP requests. The concern is that many "open" environments such as educational institutes may fall into this category.

Summary: likely not a threat.

G. HTTP

While the typical operation of an HTTP session is to transfer data items between the client and the server, HTTP proxy caches provide a way that an HTTP client can manipulate a server into initiating a connection to a victim web server. Most proxies will happily attempt to fetch whatever URL you request from them. These fetches look to the victim like legitimate requests; it cannot filter them out without losing all of its legitimate clients, too.

There are three limitations/defenses against proxy reflector attacks. First, it is not clear that there are enough proxy caches (as opposed to Web servers themselves) to constitute a truly large pool of possible reflectors, though with the rise of content distribution networks (CDNs) this may change (and, as noted above, even a fairly modest number of reflectors can still serve well to complicate traceback).

Second, in principle proxies can be configured to only serve a particular set of clients. However, CDN proxies likely cannot do any such restricting, because by their nature they're meant to serve the Internet public at large. On the other hand, the proxies could be configured to only serve the pages of their customers. Anecdotally, they do not appear today to have this restriction.

Third, the connection between the slave and the reflector cannot be spoofed (unless the reflecting proxy has predictable sequence numbers), and hence monitoring or logging at the proxy will identify the slave's location.

This last is a major shortcoming. It means that the attack might be quickly traced back—all it requires to expose the slave is one alert administrator among the many off of which a slave is reflecting.

Summary: would be a significant threat were it not for the likely quick traceback due to the non-spoofed connection from the slave to the proxy. Definitely a significant threat if servers running on stacks with predictable sequence numbers are widely deployed.

H. Other TCP applications / Gnutella

There are a vast number of different TCP-based applications, and certainly some of them will provide some form of relaying, implicit or otherwise, that can be exploited by an attacker to serve as a reflector (e.g., SMTP relays [Po82]; FTP servers and PORT directives [PR85]).

For nearly all of these, however, the same limitation applies as stated above for HTTP reflectors: triggering the reflection requires a non-spoofed connection from the slave to the reflector, which then exposes the slave to traceback.

An exception, however, is Gnutella [Gn00]. As explained in [Be00b], Gnutella includes a "push" facility analogous to an FTP PORT directive that instructs the server to connect to a given IP address and port in order to deliver the Gnutella item. However, the key difference between this form of reflection and that for FTP is that the Gnutella "push" directive can first propagate through the Gnutella network, becoming separated from the client (in our case, the slave) that injected the request. Thus, while the victim can readily trace back to the Gnutella server that is attempting to connect to the victim, the next step of tracing back to the slave is essentially impossible: the request has lost its origin, and there is no information that the Gnutella server can log, other than its immediate neighbor who passed along the request. While in principle with enough logging one could trace back the chain from neighbor to neighbor to (eventually) the requesting client, it seems certain that this will prove administratively impossible. The only apparent fix would be to modify the protocol to include propagation path information with "push" directives.

Finally, other large overlay networks (IRC, distributed games) may have similar functionality that can be exploited.

Summary: Gnutella could be a major problem.

I. Other UDP applications

To our knowledge, there are no other UDP applications sufficiently widespread to serve as a major potential pool of reflectors. If there were, however, and they did not reside on a well-known port (such as UDP port 19 for *chargen* [RP94]), then they could be used to attack UDP-based victim servers such as DNS servers by forging the victim's source address and well-known port. While the reflection generated by the application would be a junk request as far as the victim server was concerned, unless the request had a set of characteristics that permitted filtering it out, the victim would have to spend resources determining that the request was indeed invalid, and the attack would be effective.

Summary: while UDP applications could be a threat in principle, no immediate threat is apparent.

IV. Implications of reflector attacks for traceback

A major advantage to attackers in using reflectors in DDOS attacks is the degree to which they complicate traceback. First, instead of the victim being able to trace back the attack traffic from themselves directly to the slave, they must induce the operator of one of the reflector sites to do so on their behalf, which can be administratively cumbersome or difficult. Furthermore, if that traceback is then done using a scheme that relies on observing a high volume of spoofed traffic, such as ITRACE or probabilistic packet marking, then the attacker can undermine the traceback by spreading each slave's trigger traffic across many reflectors, greatly increasing the amount of time required by the traceback scheme to gather sufficient traffic to analyze.

However, if traceback is done using a scheme that also works for low volume flows, such as SPIE, then this advantage disappears, and the attacker should *not* spread out each slave's trigger traffic, as doing so will increase the chances that the slave will be detected by one of the different cooperating operators.

Another facet of the analysis in the previous section to keep in mind is that some forms of reflector attacks require legitimate (non-spoofed) connections from the slave to the reflector, such as exploiting HTTP proxies. Such reflector attacks will expose the slave to potentially immediate traceback.

For reflectors running on TCP stacks with guessable sequence numbers, the attacker may well be able to establish the necessary slave-to-reflector connection without exposing the slave's IP address; however, guessing sequence numbers generally requires establishing a series of legitimate connections beforehand, in order to infer the pattern of sequence number generation. If the logs at the reflector include these initial probes, then the slave may still be exposed. That said, for application-level logs, the attacker may be able to escape having the probes logged by failing to complete the 3-way TCP connection establishment handshake, in which case the application running at user level will generally never see the connection.

Finally, we note that Barros independently discovered DDOS reflector attacks, and proposed an elegant modification to ITRACE to address them [Ba00]. Barros' refinement is for ITRACE routers to sometimes send the ICMP message to the *source* of the just-processed packet rather than its destination. The net effect is that if a slave is forging traffic from a victim in order to dupe a server into acting as a reflector, occasionally routers *on the path between the slave and the reflector* will send ITRACE messages to the victim, enabling the victim to trace back the attack to the slave(s).

Note that the efficacy of Barros' "reverse ITRACE" mechanism does not depend on N_r, the number of reflectors, but only on N_s, the number of slaves. From our analysis above, it is clear that this appealing scaling property makes the mechanism helpful for defending against many forms of reflector attacks.

V. Summary

The above analysis indicates that there are several significant reflector attack threats:

- Victims must be able to cope with loss of general access to remote services, due to the need to filter out SYN ACKs. Such filters could, however, include holes to allow access to a small number of particular remote servers.
- DNS servers can be attacked by reflectors serving recursive queries. Damage is limited only by the size of the reflector pool, i.e., how many name servers there are that support recursion and accept requests from arbitrary clients.
- TCP-based servers are for the most part somewhat protected against application-level reflection *assuming* that enough of the application servers keep sufficient logs that the non-spoofed connection between the slave and the re-

flector can be used to trace back the attack to the slave. Without this assumption, Web servers can be attacked by requests chained through proxies serving as reflectors, SMTP servers by mail sent through relays, and any TCP-based server by requests reflected through mechanisms such as FTP PORT directives.

• TCP-based servers running on TCP stacks with guessable sequence numbers are a severe threat. Not only do they allow application-level reflection without easy identification of the slave (unless the precursor traffic probing the sequence-number progression is logged), but they also can provide major amplification of the attack traffic due to the use of ACK-splitting techniques [SCWA99].

• Gnutella's "push" facility appears to be a significant threat.

VI. Acknowledgments

My thanks to Steve Bellovin and Cesar Eduardo Barros for their valuable comments on this work, and also to the anonymous reviewers for their excellent comments.

References

[Ba00] C. Barros, "[LONG] A Proposal for ICMP Traceback Messages," `http://www.research.att.com/lists/ietf-itrace/2000/09/msg00044.html`, Sept. 18, 2000.

[Be96] S. Bellovin, "Defending Against Sequence Number Attacks," RFC 1948, May 1996.

[Be00a] S. Bellovin, "ICMP Traceback Messages," Internet Draft, `http://www.research.att.com/~smb/papers/draft-bellovin-itrace-00.txt`, March 2000.

[Be00b] S. Bellovin, "Security Aspects of Napster and Gnutella," `http://www.research.att.com/~smb/talks/NapsterGnutella/index.htm`, June 2000.

[Br94] R. Braden, "T/TCP — TCP Extensions for Transactions: Functional Specification," RFC 1644, July 1994.

[CFSD90] J. Case, M. Fedor, M. Schoffstall and C. Davin, "Simple Network Management Protocol (SNMP)," RFC 1157, May 1990.

[Ce97] CERT Coordination Center, "FTP Bounce," CERT Advisory CA-1997-27, `http://www.cert.org/advisories/CA-1997-27.html`, December 1997.

[F+00] D. Farinacci, T. Li, S. Hanks, D. Meyer and P. Traina, "Generic Routing Encapsulation (GRE), RFC 2784, March 2000.

[FS00] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2827, May 2000.

[Gn00] Gnutella, http://gnutella.wego.com, 2000.

[Gu01] Guardent, "Guardent releases information regarding flaw in Internet infrastructure," `http://www.guardent.com/pr2001-03-12-ips.html`, March 2001.

[Me00] P. Metzger, private communication, February 2000.

[Mo87] P. Mockapetris, "Domain names — implementation and specification," RFC 1035, November 1987.

[Mo90] J. Mogul and S. Deering, "Path MTU discovery," RFC 1191, November 1990.

[NBBB98] K. Nichols, S. Blake, F. Baker and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC 2474, December 1998.

[Pa99] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks* 31(23–24), pp. 2435–2463, December 1999.

[Po80] J. Postel, "User Datagram Protocol," RFC 768, August 1980.

[Po81a] J. Postel, "Internet Protocol," RFC 791, September 1981.

[Po81b] J. Postel, "Internet Control Message Protocol," RFC 792, September 1981.

[Po81c] J. Postel, "Transmission Control Protocol," RFC 793, September 1981.

[Po82] J. Postel, "Simple Mail Transfer Protocol," RFC 821, August 1982.

[PR85] J. Postel and J. Reynolds, "File Transfer Protocol," RFC 959, October 1985.

[RP94] J. Reynolds and J. Postel, "Assigned Numbers," RFC 1700, October 1994.

[SCWA99] S. Savage, N. Cardwell, D. Wetherall and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver," *Computer Communication Review*, 29(5), pp. 71–78, October 1999.

[SWKA00] S. Savage, D. Wetherall, A. Karlin and T. Anderson, "Practical Network Support for IP Traceback," *Proc. ACM/SIGCOMM*, pp. 295–306, August 2000.

[S+01] A. Snoeren, C. Partridge, L. Sanchez, W. Strayer, C. Jones and F. Tchakountio, "Hash-Based IP Traceback," *Proc. ACM/SIGCOMM*, to appear, August 2001.

[SP01] D. Song and A. Perrig, "Advanced and Authenticated Marking Schemes for IP Traceback," *Proc. IEEE INFOCOM*, April 2001.