Steps Towards a DoS-resistant Internet Architecture

Mark Handley, Adam Greenhalgh
University College London
{M.Handley, A.Greenhalgh}@cs.ucl.ac.uk

ABSTRACT

Defending against DoS attacks is extremely difficult; effective solutions probably require significant changes to the Internet architecture. We present a series of architectural changes aimed at preventing most flooding DoS attacks, and making the remaining attacks easier to defend against. The goal is to stimulate a debate on trade-offs between the flexibility needed for future Internet evolution and the need to be robust to attack.

Categories and Subject Descriptors

C.2.0 [**COMPUTER-COMMUNICATION NETWORKS**]: General - Security and Protection

General Terms

Design, Security

Keywords

Internet, Denial-of-Service, Security, Network Architecture

1. INTRODUCTION

Denial-of-Service (DoS) attacks are one of the most significant problems currently facing the Internet and its users. For many users these attacks are merely an irritation - even if they understand the reason for the poor performance they occasionally observe. However, for the Internet to achieve its full potential, it has to be able to offer highly reliable service, even in the face of hostility.

We will examine some significant changes to the Internet architecture aimed at making the Internet more robust. Such changes should not be made lightly - any widespread change has real costs associated with it. In writing this paper we are only too aware that the problem of DoS attacks can not be *completely* solved by the architecture we propose. The problem needs to be tackled on many fronts simultaneously. However we do believe that architectural changes are necessary. The only question is what form those changes must take? In this paper we will take a fairly radical position, with the aim of stimulating this debate.

SIGCOMM'04 Workshops, Aug. 30+Sept. 3, 2004, Portland, Oregon, USA.

2. THE NATURE OF THE PROBLEM

The first step in considering a security problem is to consider the nature of the threat. In [9], the Internet Architecture Board provides a detailed discussion of the nature of DoS attacks on Internet systems, and we strongly recommend this document. Basically all systems are vulnerable to some form of attack, be they clients, servers, firewalls, routers or links. Attacks can attempt to exhaust processing power, memory, bandwidth, quotas, disk-space, and pretty much any other "consumable" that a system requires to perform its job.

One modern PC connected to a high-speed network can source around 1Gb/s of traffic, which is enough to saturate many network links and, if the traffic is carefully crafted, enough to overload many large servers. However, traffic from a single machine is relatively easily filtered. Although automated mechanisms to *push-back* such filters towards the source are not widely deployed, there are few technical problems in doing so[13][10].

Unfortunately *source-address spoofing* makes it harder to push-back filters without causing collateral damage. Further, many DoS attacks are *reflection* attacks[16], where the attacker sends traffic to a third party, spoofing the source address of the victim. The third party then replies to the victim, overwhelming them. The attacker can then use many third parties to spread his attack, so now the traffic is "originating" from all over the Internet. In addition, some reflection attacks manage to *amplify* the original attack because the responses sent by the third party are larger or more numerous[6][9] than the original messages sent by the attacker.

The biggest DoS problem is caused by *distributed* denial of service (DDoS) attacks, where the attacker compromises a large number of systems and then uses these "zombie" systems to attack the victim. DDoS attacks of sufficient scale provide the firepower needed to overwhelm almost all victims. They can also be combined with spoofing or reflection to make the attack even more difficult to defend against. Currently most DDoS attacks do not bother to spoof the source addresses because, as no automatic push-back mechanism is widely deployed, it takes so long to shut down each zombie that there is no need to hide their identity.

DDoS is principally an issue due to widespread exploitation of software vulnerabilities, which permit the control of large numbers of compromised systems. To gain sufficient scale, such exploitation is typically automated using worms, viruses, or automated scanning from already-compromised hosts (so called "bots"). Fast-spreading worms are extremely hard to combat in the current Internet Architecture, so these are a particular concern[14]. Although viruses and bots are a serious issue, their spread rate is slower, which permits a wider range of defense options.

3. DEFENSE

It is important to begin by recognizing that it is not possible to completely protect all servers against all DDoS attacks. If a sufficiently subtle attacker with sufficiently many compromised hosts at his disposal can mimic legitimate traffic well enough that the victim cannot tell good from bad, there is little that can be done beyond load-shedding, adding server resources, and minimizing collateral damage. However, our ultimate goal is that this is the *only* way to persistently DoS-attack a server, and that routers and client systems are invulnerable to DoS attacks.

Viewing the problem from a high level, there are many tracks we can take to make a network architecture that is more resilient to DoS than the current Internet:

- Significant improvements in end-system software security will reduce the ease with which systems are compromised, and hence make the construction of zombie-armies more difficult. However, we do not expect this to be sufficient by itself.
- Reducing the ability of worms and viruses to spread quickly will reduce the threat of very large scale DoS attacks. Fast spreading worms are a particular threat, because they outpace the speed of any possible human mediated response.
- Preventing source-address spoofing will aid the shutdown of attacks that do occur using push-back mechanisms.
- Preventing reflection attacks will also aid the shutdown of attacks and prevent innocent third parties from being implicated or harmed by reactive defenses.
- Without source-address spoofing and reflection attacks, automated push-back mechanisms would be much more accurate and effective.
- Large-scale wide-area distribution of key services such as DNS would localize attacks, reducing the attacker's advantage and minimizing collateral damage.
- Hosts not wishing to receive incoming connections attempts from the public Internet should not have do so.
- Router-to-router traffic should be effectively isolated from all other traffic to reduce DoS threats to routers. Although this is at least as important as the points above, it is not the main focus of this paper, so we will not discuss it further.

In this paper we do not discuss the protocol and implementation *details* needed to ensure that transport protocols and applications are robust to DoS. Instead we concentrate on architectural changes that more widely restrict an attacker's freedom and permit more effective defense.

3.1 Advertised Service

In the Internet architecture, a route advertisement is effectively a service advertisement for all the hosts on that subnet, saying "route packets to these hosts". However, such a service advertisement is rather more liberal than is generally desired. What is mostly desired is the ability to route service requests to a server for that service, and to route responses back to the client *and nothing else*. This is of course an over-generalization, but we shall use this as a starting point for a revised service model. Such a service model would have immediate implications for DoS:

- A client could not send any request to another client, which prevents a whole category of DoS attack, and also prevents client-to-client worms.
- If the server to which a client initiates communication is the only host that can send packets back to that client, then no other server can DoS that client. This also prevents many reflection attacks.

4. TOWARDS A DOS-RESISTANT ARCHITECTURE

How might we implement such desired restrictions without encountering scalability issues? In this section we will propose a set of architectural changes that, taken together would greatly limit the scope for attack. The aim is to allow all the desired modes of interaction between systems, but greatly restrict everything else.

Step 1: Separate Client and Server Addresses

The IP address space[1] can be divided into a set of client addresses and a set of server addresses. The aim is to allow clients to initiate connections to servers, but not to allow clients to initiate connections to clients, nor servers to initiate connections to servers.

The benefits of this depend on where the restriction is enforced. Even if it were only enforced by the recipient, this would immediately reduce the threat from worms. A worm would have to spread from client→server→client, exploiting two separate vulnerabilities. This greatly slows the worm because the server→client phase depends on clients choosing to contact an infected server. Very fast spreading worms would no longer be possible. Worms that attempt to spread via contagion are still possible, but are significantly more likely to be spotted in the client→server phase in time to react by the sort of organized honeypots proposed in [20].[2]

In addition, many reflection DoS attacks on servers (such as bang.c[9]) are prevented. A reflection attack on a server would require server→client→server communication, and most clients are not going to respond to a new connection request from a server[3].

In the current Internet, some of these benefits arise from the use of Network Address Translators. However, NATs only benefit those clients who chose to use them; attackers simply choose to attack from hosts that are not behind NATs. By formalizing the asymmetry and accepting it as a key part of the architecture, the benefits can be much more widespread, as we will show below. In addition the downside of NATs not being a consistent part of the architecture can be avoided.

Clearly if many hosts have both client and server addresses then some of the benefits are lost; we hope that few hosts will need both *globally-reachable* client and globally-reachable server addresses. Typically a server should not be initiating wide-area outgoing connections, and most clients only need to accept incoming connections to permit local management. Obvious exceptions are peer-to-peer applications, and telephony-style applications. We will discuss these special cases in Section 5.1.

Step 2: Non-global Client Addresses

A client address does not need to have any global significance. It only needs to have significance along the path between the client and the server, so that packets from the server can be returned to the client. In fact, a client *wants* its address to not have global significance - this prevents distributed DoS attacks on the client

[1] Typically we are thinking about IPv6 addresses here due to the additional flexibility available from longer addresses.

[2] The state of the art in honeypots still needs to advance somewhat before this could be considered a solved problem.

[3] The exceptions would be stateless reflection such as responding to an ICMP echo request.

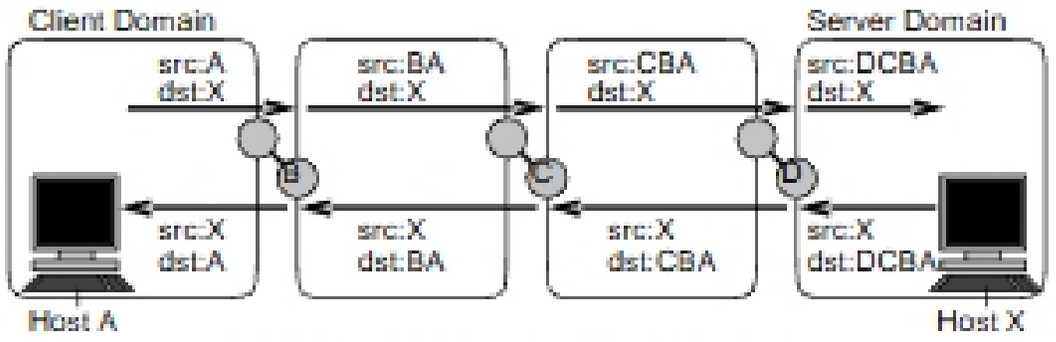

FIGURE 1—Path-based Addressing

or its access network unless each attacking host can figure out a workable address to route traffic towards the client.

One way to allocate client addresses in a non-global manner would be for the source address to be constructed domain-by-domain as the packets travel from client to server. This is illustrated in figure 1. On entry to a domain, a local routing ID (indicating the next domain toward the client) would be pre-pended to the source address. Packets being forwarded back towards the client would then be forwarded by the usual longest-prefix-match forwarding mechanisms within each domain (in this case the prefix is the local ID for the next domain). As the packet leaves each domain heading back towards the client, the local ID is removed.

It is likely that such address prepending can be done at line-speed in some of todays backbone routers with only firmware upgrades.

The goal is that even if one malicious server in the Internet knows a client's address, no-one elsewhere in the Internet who is given that address can easily deduce a workable client address to reach the client from their location. The simple prepending scheme above provides this protection to some degree, but it is possible that more security is required. This can be achieved at the expense of additional forwarding cost by encrypting the client address before prepending the local ID in the client-to-server path, and decrypting it in the return path after removing the local ID. For example, the router with interface C in figure 1 would receive client→server packets with source address BA and forward then on with source address Cf(BA) where $f(x)$ is a shared key encryption function[4]. Whether such additional security is really necessary is an open question, but the architecture would permit it as a matter of local policy. In practice it is likely that only a few core ISPs need to do this to gain most of the benefits.

Such path-based client addresses have some useful properties:

- The make complete source-address spoofing impossible for clients. A client address will always reveal the path back to the origin domain, although it may not reveal the precise host at that domain. This should allow pushback mechanisms to function effectively against traffic sourced by clients.

- In contrast to the current Internet, the path between client and server would be symmetric at the domain level (although not at the router level). The economic implications of this are not completely clear to us. However, at the very least it should simplify many network monitoring functions for transit ISPs.

  Perhaps more importantly, if an ISP monitors that there is a large amount of traffic in one direction but no appropriate responses in the reverse path, then the ISP can conclude by itself that this traffic is malicious. This may allow link-saturation attacks to be identified and shut down. However, such a deduction may require collating information from multiple routers within the ISP's domain, so may not be easy without substantially improved network monitoring tools.

[4] All the border routers of a domain share the same key

- All reflection attacks against remote client targets are prevented. A reflection attack would need to go client→server→client, but path-based addresses do not allow the spoofing of a remote victim's client address.

- Many routing DoS attacks on client systems, such as announcing bogus routes, are prevented as client routes are simply not announced inter-domain.

It is important to note that such client addresses are inherently changeable. If the client moves location, or the inter-domain routing between client and server changes, then the client's address as seen by the server will change. This mechanism therefore requires that transport connections have access to a more stable ID above the IP layer. Proposals such as HIP[15] would provide such a suitable ID, although other simpler forms of ID might also suffice.

Another implication for transport protocols becomes clear if we consider what happens when routing changes in such a way that the original server→client inter-domain path becomes unusable. The server has no way to reach the client until it next receives a packet from the client indicating a new valid path. This is primarily a problem if a connection is idle when a change occurs, and the server then wishes to re-start communication. Periodic "keepalives" are one way to solve this. Alternatively a client can passively monitor the inter-domain route for this server and deduce when it needs to send a keepalive to update the server's knowledge of the its address. To do this a client would need access to the BGP AS Path for the route used to reach the server. For many practical reasons with BGP as it is currently deployed, this is not quite as simple as it might seem, but the general idea is feasible.

Step 3: RPF Checking of Server Addresses

Using path-based client addresses severely restricts source-address spoofing by a client, but it does not restrict spoofing by servers. However, the domain-level symmetry that emerges from using client-based addresses means that packets traveling from server→client follow the reverse client→server inter-domain path. The allows domain boundary routers to perform a reverse-path forwarding (RPF) check on the source address of server→client packets. This check largely prevents a server from spoofing the address of a server in a different domain.

When combined with path-based client-addresses, one effect of this is to make it much harder to launch blind DoS attacks on ongoing communications, such as injecting a TCP Reset into a connection whose existence can be inferred.

Step 4: State Setup Bit

Not all packets are equal. Packets that require the recipient to set up new state are more risky from a DoS point of view than those that don't. It is useful to single these packets out for special handling in a manner that is independent of the upper layer protocol.

Packets that will cause transport communication state to be set up (especially connection setup) should set a new state-setup bit in the IP header. Other packets would leave this bit unset. This serves a number of purposes:

- It provides a generic protocol-independent way to identify packets that need special validation. A server receiving a connection setup request with this bit not set would simply discard the packet.

- Stateful firewalls can validate packets with this bit set before instantiating state. We will discuss what form this validation might take below.