Concordia University College of Alberta

Master of Information Systems Security Management (MISSM) Program

7128 Ada Boulevard, Edmonton, AB

Canada T5B 4E4

Reverse Discovery of Packet Flooding Hosts with Defense Mechanisms

by

MURRAY, Brian

A research paper submitted in partial fulfillment of the requirements for the degree of

Master of Information Systems Security Management

Research advisors:

Pavol Zavarsky, Director of Research and Associate Professor, MISSM

Dale Lindskog, Assistant Professor, MISSM

Reverse Discovery of Packet Flooding Hosts with Defense Mechanisms

by

MURRAY, Brian

Research advisors:

Pavol Zavarsky, Director of Research and Associate Professor, MISSM

Dale Lindskog, Assistant Professor, MISSM

Concordia University College of Alberta
Department of Information Systems Security
7128 Ada Boulevard, Edmonton, AB T5B 4E4, Canada

Reverse Discovery of Packet Flooding Hosts With Defense Mechanisms

Brian Murray
brian@game-sat.com

Reverse Discovery of Packet Flooding Hosts With Defense Mechanisms

Abstract - Distributed Denial of Service attacks have long been a staple of the Internet's malware community that has not been dealt with outside of companies with large budgets. Many of these attacks use spoofing to obfuscate their originating address. It is this problem that I intend to address with a new distributed traceback approach. In essence, it removes the effectiveness of clusters of infected botnet computers by 'worming' its way backwards through the Internet to locate the source of packets destined for a host under attack. Once the source is discovered, countermeasures are deployed at the source location, which ultimately prevent the traffic from ever entering the Internet, and decreasing the overall demand on the Internet. This is a benefit for both Internet backbone providers, as well as customers to these Internet service providers in that they would no longer be forced to suffer from extended periods of Internet outages.

Introduction

Denial of Service (DoS) attacks are still a widespread problem on the Internet, with 25% of companies suffering from an attack [1]. There are several different attack types that could be considered a DoS [2]. They include Protocol Feature attacks, Data Flood, and Application Level attacks. The solution presented in this paper explicitly addresses Data Flood type attacks, but could extend to other types of attacks with simple modifications. Attack Countermeasures can be classified into 4 categories: prevention, detection, mitigation, response [2]. The method proposed would fall into the mitigation and response categories. Deployment locations can be classified as victim, intermediate, and source network. This solution provides the ideal scenario where the detection happens at the victim network, but the defense mechanisms are deployed at the source network, with very little overhead at the intermediate network.

Denial of Service Attacks

A DoS attack is a type of attack that prevents normal, legitimate traffic from being served. The most common way to do that on the current Internet is to flood a large amount of traffic at a specific host. The victim is then unable to handle the excess traffic, and will mistakenly discard legitimate traffic. While this may not be an issue for a web photo gallery, it does pose a significant problem for a small business that is just starting out on the Internet. When millions of dollars of business gets conducted every hour, a website cannot afford to be down for any significant period of time. Older attacks have abused the very systems of the Internet that allow it to function, such as a TCP SYN flood attack. However, these low volume attacks have mostly been solved with some ingenuity of Internet architects, such as with TCP syncookies. Brute flooding, however, is a very simple problem, but also very troublesome. The number of compromised hosts on the Internet is growing every day, making large botnets even bigger than they already are. Unfortunately, there is no simple way to stop these attacks, so a more complex approach is required.

There are two standard flood type DoS attacks. The first is a standard DoS, where a single host floods another host. This means the attacker must have equal, or greater bandwidth than its target. The second, and more dangerous attack, is a Distributed Denial of Service (DDoS) attack. This is identical to a standard DoS attack, except there is many hosts attacking a single target. Usually these hosts are linked and controlled by a single hacker, or group of hackers. Since there are many hosts sending data, only their aggregate traffic needs to be greater than what the target can handle. That means, the more hosts that are attacking, the less traffic needs to be generated by each attacking host. Since botnets can contain thousands of hosts, each attacking host may only need to send very small amounts of traffic.

One simple fact always remains in flooding DoS attacks: the target is overloaded with inbound packets. Whether an application cannot handle the load, or an actual network interface is forced to drop packets as its queue is full, no packets may reliably reach the attacked network or host while an attack is taking place. Upstream traffic, however, is controlled completely by the attacked host. This provides a convenient path to notify upstream routers of the issue.

Current Protection Systems

There are already a few approaches to preventing such an attack. However, for the most part, they are ineffective at stopping large attacks, or attacks on smaller hosts with fewer resources than a large corporation.

The first type of existing scheme uses an outside party to filter traffic. All Internet Domain Naming System (DNS) records point to this third party, who has a large amount of bandwidth. Traffic destined for the site is routed to this third party. Then, all the traffic they deem legitimate gets sent on to the real servers. This can create a fairly large lag time, and can reduce the responsiveness of your site. As a result, customers may be turned off from doing business with you. Also, there is no guarantee that the third party can handle the attack. At best, this can stop small or medium sized attacks, but is still prey to large attacks.

The second type, offered by Cisco, is very similar in that it filters bad traffic. It involves an ISP having a second backbone that filters the traffic. Since this second “clean pipe” is already in line with the existing network connections, it does not suffer from the extra hops as the other solutions do. [3] However, a large DDoS attack can still take down one of these massive Internet backbone connections.

A third solution, which is probably the most widely known, is called ‘Anycast’. It is primarily used by the DNS root servers, which are critical to the functioning of the Internet. Anycasting works by redirecting traffic to a local provider, instead of a far reaching provider. That is to say, you request a DNS resolution from a server on the other side of the country, but the request is actually handled by a nearby server. This is usually accomplished by providing multiple BGP routes to the same Autonomous System (AS). Each route terminates at a different location, which is more local than another route. Since the servers are geographically diverse, no attack can take down the entire DNS structure. Every root cluster may only serve a small portion of the Internet, instead of all of it. So, if a large scale attack was to originate from Singapore, their attack traffic would never touch the North American root servers as the attack packets would only be delivered to the local Singapore DNS root servers. For Anycast to be deployed for a specific company, they must have the money to deploy in several geographically diverse data centers. Their routes must also remain highly stable for any connection oriented traffic to function. Their data centers must also interact and collaborate data between them. For a small business, Anycast is not a viable solution.

Recently, a lot of research has gone into solving the problem, with many different approaches. Some use overlay networks to manage the attack routes [4] [5]. These merely obfuscate the traffic path. A smart attacker will merely flood out the overlay network from a variety of different hosts.

The primary concern with an overlay network is that it is a second set of routes. Some are merely the same routes, but with tunnels over top that create a second virtual network. These secondary routes need to be maintained in order for the system to work, even if they are established temporarily. Some require authentication of connecting hosts to use the overlay network. This allows them to do rudimentary filtering of traffic. However, an attacker can still flood out the overlay network's entry points with data, negating the whole system. Even overlay networks that have alternate physical paths are still vulnerable, as an attacker with an inordinate amount of attacking hosts can still overcome the bandwidth allowed by the second network. The second approach mentioned in this section, which is offered by Cisco, is a type of overlay network.

Other systems use traceback methods, such as the ANT system, where “ants” crawl up the network. The ANT system is based on the natural path creating method that ants use in the wild. Ants march towards a destination, each laying a trail of scent. As more ants lay a trail, the scent gets stronger. The path with the strongest scent is the path that more ants follow, and subsequently strengthen the path even further. In the ANT system, the “ant” lays a trail along the path with the most traffic [6]. Most traceback systems that work by communicating with peers, such as ANT, require updates to a vast number of routers in the chain. Other types of systems proactively tag packets for the destination to parse when it is received, so that it gets a better idea of where the packet originates from [7]. However, these proactive systems add significant overhead to intermediary routers, as each needs to add its tag on to every packet.

Proposed new solution

The proposed solution requires 3 tiers of development. The basics of this protocol are quite elementary. First, an attack needs to be detected. This can be done automatically, such as with many currently deployed Intrusion Detection Systems (IDS), or manually with administrator intervention. Next, the attacker must be discovered. For this, a reverse path discovery method is used. Finally, the attacker must be stopped.

This last stage only requires the closest trusted routers to the attacker to prevent data flow.

The key component to this new method is the reverse path discovery. Since the only reliable piece of information in an attack is the destination, the protocol uses that destination address to detect the source of the attack. When an attack is detected on a client system, the client system sends an initiator packet to its nearest router. That edge router then sets all of its participating interfaces, with the exception of the interface that received the initiator packet, into a listening mode. In the listening mode, the interface is scanning for incoming packets that match the destination host or network that was identified in the initiator packet. When a packet is detected on the participating interface, the interface then stops listening, and performs one of two tasks. It may deploy a countermeasure by blocking packets to the attacked network, or by filtering the packets with deep packet inspection that are destined for the attacked network. This would be most common when packets are detected from a stub network, such as on an edge router. The interface may instead notify its peers. It can do this either with unicast to each peer, or via multicast to notify a wide range of peers. The interface notifies its peers by using the same initiator packet that was sent to it, but replaces the authentication data with its own authentication data, as is discussed later. This process is repeated until no packets are detected on interfaces that are set to notify peers.

Once the source has been narrowed down, the attack must be stopped. The hardest part is detecting attack packets versus legitimate packets. However, since the destination is already known, the differentiation is much easier, since an attack on that specific host can be verified. That is, it limits the exposure to false positives, since no detection is done until an attack has been verified by the destination.

This system allows for every company, small or large, to mitigate DoS attacks. It is deployed by ISPs, but authenticated access is granted to smaller companies. This means that no small company has to make a large investment in redundant data centers with multiple BGP advertised routes. Since the notification packets only travel upstream, no two way communication is required while an attack is under way.

In the first two schemes that I have mentioned, the solution is merely to add more bandwidth, but at two different locations. Unlike those schemes, the proposed solution uses existing resources to mitigate the attack. It does not rely on the attacker having less than sufficient traffic to overload the extra bandwidth added. The proposed solution also allows countermeasures to be deployed closer to the attacker, which means that less Internet bandwidth is used, which is a benefit to the ISP. Ultimately, this also means that the countermeasures are only checking much smaller amounts of bandwidth, and is therefore significantly more efficient at separating the legitimate traffic from the malicious traffic.

The ANT system, at a glance, looks like a very good system as it mimics nature. However, nature and evolution work on a “just good enough” principal, as opposed to a “perfection” principal. The ant path creating system works fine in nature for its purpose, but when discovering an attacker on a network, the actual path used is unimportant. The only important piece of data is the actual source of the attack. In nature, the trail is used later to aid in navigation for future ants. On the Internet, the destination address can be reached using existing routing resources, so long as the attackers location is known and translated to the proper source address of the attacker. Laying a trail merely wastes valuable resources on already strained Internet routers.

A variety of tagging systems also suffer from the same issue. They put a large weight on discovering the actual path of the attack, when the only thing needed is the actual attackers source. If an attack was originated only from a small area of the Internet, these systems would work very well, as traffic could merely be blocked at the first common router to the attackers. However, attacks come from very diverse areas of the Internet, so the attack would either be blocked close to the destination, or many blocks would need to be placed throughout the Internet, making the tagging mostly unused.

On top of intermediate tagging being highly wasteful, the issue remains of how do you stop the attack. These systems would require a separate protocol to inform the attackers nearest router to stop the attack. The proposed solution does not concern itself with intermediary hops, and is capable of deploying countermeasures at the instant the attacker is discovered. There is also no processing of the attackers packets at any routers, nor is the attacked host required to reconstruct the attack path. Once the attacker is discovered in the proposed solution, a countermeasure can be deployed instantly, without the need for a second notification to inform the attackers nearest router that an attack is taking place. As a side note to a second protocol being needed, every edge router on the Internet would need to allow for authentication of every host that could be attacked. In the proposed solution, the only direct peers or peers within its multicast group need to be authenticated. This means that an initiator can be started from a host in Toronto, and each intermediary only needs to trust the validity of each adjacent hop, with the router closest to the attacker only needing to trust the next nearest router, as opposed to trusting the host in Toronto.

Protection Method Development

There are several factors that need to be considered when creating a new algorithm. Many problems need to be solved so that the final product is highly efficient.

Communications Protocols

For multiple routers to interact, they need to have an efficient communication protocol. However, delivering the message can be as difficult as designing the message itself. The message cannot be sent to routers that don’t need the message, but must make it to all routers that do need the message. This is why blanketing out a request to all hosts may not be a good idea.

Unicast

Unicast is the most common protocol in use on the Internet. All traffic between any two hosts is unicast, such as web traffic, DNS traffic, or Email.

To use unicast with the proposed solution, each router in the chain needs to know about all other routers near it. While this may not be a problem on a small scale, it could end up being a very large problem as a single router may have a dozen peers. Also, this would require that all routers in the path between the attacker and the victim would need to be updated to support this new method.

Broadcast

Broadcast traffic is different from unicast in that everyone on the local 'broadcast domain' hears the message. Broadcast, however, is only significant to the local network. It can be directed to a specific subnet, but that is often disabled to prevent abuse. Since only locally significant broadcast can be used, only other routers directly connected to each router can hear the message. This alleviates most of the need for each router knowing its counterparts. One broadcast message will inform all of its peers about the attack, and have them shut down the traffic flow and notify upstream. However, since only the next router can hear the message, it means that every router in the chain still needs to be patched with support. This is a slightly better protocol, but still not good enough.

Multicast

Multicast is still not widely used outside of routing protocols, even though it is well supported. A multicast packet can be routed, and even "split" amongst destinations. That is, one packet may be sent, but many destinations may receive the message.

There are two methods for distributing multicast traffic. The first is called a source tree. Source trees are sometimes referred to as shortest path trees (SPT), since they represent the least hops between the source and destination. Using this method, every router between the source and destination needs to maintain a table of each route. This can lead to very large memory requirements.

The second method is called a shared tree. All multicast traffic is sent to a rendezvous point (RP), then routed down the multicast tree to all listeners. With this, only one route is required. The memory requirements are significantly less.

Each of these two methods have their own drawbacks. Obviously, the more routers there are on the network, the more routes would be required for the first method. This number could get quite large within a single AS. However, the second method provides a single point of failure. If an attacker knew the RP, they could attack the RP as well as the destination, removing the effectiveness of the protocol. This could be mitigated by moving the RP frequently, and ensuring few people knew where the current RP was located.

Since every router may be able to route the message without understanding it, most routers on the chain will have no need to be patched or updated to support this. It also means that no routers need to know where its peers are. Ultimately, this is the best solution as it allows for much more rapid dissemination of information, as well as requires the least updates from intermediary routers. Without every router needing updates, a perimeter can be extended very quickly.

For example, if we had two cities, like Edmonton and Calgary. A host in Edmonton is being attacked, so it requests action be taken by its nearest router. The router then notifies the Edmonton multicast group. All access routers and intercity routers within Edmonton are instantly notified via multicast from a single request by the router. If flooding is occurring from an access router, and if blocking is deployed as the countermeasure, the traffic is purely blocked. If traffic is seen from a intercity router coming from Calgary, then the Calgary mutlicast group is notified by the Edmonton-Calgary intercity routers. All access routers within Calgary are then checked for attack sources as well. No other routers are required to have updates, and merely route the multicast traffic. If every router between the source and destination needed to be notified, then the response time would be greatly increased.

To quantify the benefit of multicast over unicast, we need to quantify each component of the notify process. First, I will denote the time between the router receiving the packet, and when it begins to notify the next hop in the chain as *P*. Next, the time between each hop between the source and destination will be denoted by *H*. The number of hops will be denoted with an *h*. Therefore, the time T_{uni} for unicast would be:

$T_{uni} = \text{h x P}$

So the time to notify a router that is 4 hops away, with processing time (P) of 1 second:

$T_{uni} = 4 \text{ x } 1$
$T_{uni} = 4$ seconds

Multicast traffic only needs the source and destination to process the information, with the rest of the routers merely passing on the data. Therefore, we are left with:

$T_{mul} = (2 \text{ x P}) + (\text{H x h})$

If we use the same numbers, and assume a hop time of 1 millisecond each using SPT:

$T_{mul} = (2 \text{ x } 1) + (0.001 \text{ x } 4)$
$T_{mul} = 2.004$ seconds

The more intermediary routers are added, the greater the difference. If we have 8 routers:

$T_{uni} = 8$ seconds
$T_{mul} = 2.008$ seconds

If we use unicast to communicate with each end router, that is several hops away, we then need to count each router that the router needs to communicate with. For this, we can use the time *H* for the time to send a packet to each router *n*.

$T_{uni} = 2\text{P} + (\text{H x n})$

With 50 peers, this number is:

$T_{uni} = 2(1) + (0.001 \text{ x } 50)$
$T_{uni} = 2.050$ seconds

This number is small, almost as small as that of unicast. However, unicast has another drawback. It requires each router to know of every other router. This means that all peers must be maintained. If a new router is added, all 50 peers must be added to it, and it to all 50 peers. Multicast does not require every other router to know of every other peer. Maintenance time on each router could be significant. If a peer is not added, especially an inter-domain transit router, it can leave a hole open to be exploited.

Software Management

The implementation of the protocol requires two separate components, the daemon and the firewall. Each has its own tasks to perform, but neither can function without the other.

Daemon

The daemon's job is to receive messages from other routers informing it of an attack. Once it receives an initiator message, it must act on it by creating firewall rules, and monitoring them for activity. Once it notices activity on one of these rules, it then must inform the next set of routers.

First, it needs to listen on a multicast address for messages from its peers. These messages will consist of the destination address and its netmask. Initiators should also contain authentication data.

Next, it needs to interact with the firewall. In the testbed system, it simply used Linux's netfilter and its 'recent' matching.

To gather information about if the rules have been matched, it will read the 'proc' nodes pertaining to the recent matches.

Lastly, it must be able to send data out to its peers, based on where the firewall rule triggered from. If the interface has no peers, it should perform some rudimentary detection tasks, such as detecting spoofed packets, then block packets that do not appear to be legitimate.

Firewall

The firewall is a very important piece of software. It must be able to dynamically add rules, while still remaining efficient. In high bandwidth environments, a null route may be used instead, but it must report that it was triggered to the daemon. In the case of the testbed, it will simply use Linux's netfilter with the 'recent' module.

In a real world deployment, the firewall rules may not even work as a normal firewall would. It may, however, establish a GRE tunnel to a IDS or other malicious traffic filtering system that is deployed by an ISP, then re-route all traffic that matches the destination through that GRE tunnel. If the IDS detects the packet is malicious, it would be dropped and logged, but if it was benign, then it would be passed along with only a delay added. The delay would only be applied while an attack is taking place, instead of always, such as with other previously mentioned methods.

Thresholds

While thresholds are not required, they can help to augment the effectiveness of the system without erroneously preventing legitimate users access to the victim. Once a threshold is reached, the router will either send out a notification to its peers out the source network interface, or will carry out an action to the packets if it has no peers out that network. The thresholds can be implemented near the attacked network, or the attacker. If they are implemented at the attacked network's router, then they could be used to trigger the reverse discovery process. One example of this would be an intrusion detection system set off by DDoS packets. If the thresholds are implemented close to the attacker, they could properly handle, and filter, all attack traffic from that subnet to the destination.

Packet Numbers

One threshold can be the number of packets received with the destination set. That is, if too many packets per second are being sent to the destination in the attack, then it could be considered as one of the attackers. This may, however, not work in the case of a low rate attack, consisting of thousands of nodes.

Connection Numbers

Some DoS attacks don't consist of raw flooding, but instead, of legitimate connections tying up useful resources on the destination server. If too many connections are attempted, it could be considered the attacker. Along with these connection numbers, the amount of data sent directly after the connection could be used. If the connection opens, but no data is sent or received, then it is likely an attack that leaves useful sockets open to handle empty connections. Adjusted window sizes could be an indication of this type of attack.

Spoofing

In the most common flooding attack, spoofing is often used. This is the purpose for this protocol. Since spoofing makes it impossible for any destination to know the source, and take action, reverse path discovery is used in this new method to find the source. Once it has gone no further than it possibly can upstream, it can check to see if any packets are being spoofed. That is, if the last router is on the access network of 30.40.50.00/24, and sees packets from 1.5.2.4, then it knows spoofing is happening, and that there is almost definitely an attacker out its interface.

In a real world implementation, a threshold that adheres to RFC 2827 [8] would be the most beneficial.

Alternate Actions

Once a threshold is met in the deployed testbed, one of two actions occur. It either blocks the traffic, or notifies its peers in an upstream direction. In a full Internet deployment, this action can be anything that the deploying ISP may desire. For example, instead of blocking traffic to the attacked network or host, it may redirect traffic from that attacking network to a classification system that determines if an attack is taking place. This could happen on overlay networks, such as those described earlier. Since few hosts would actually be filtered by each of these, there would be almost no potential for them being overloaded. This places the mitigation as close to the attacker as possible. This would allow an ISP to verify that an attack was taking place from that host, instead of relying on simple detection protocols.

Another potential action could be to simply notify the network operations group via an SNMP trap, or any notification system that the company has implemented.

Ultimately, in the real world, blocking entire non-conforming networks would not be beneficial for most cases. Instead, re-routing traffic destined for the attacked host to a deep packet inspector would be significantly more beneficial. This has the effect of requiring much less investment on packet inspectors, as most traffic won't be triggered as potential attack traffic, and as a result, less packets will be redirected to the inspector.

Security

With this type of protocol, there is a huge potential for abuse. Without security precautions, someone could spoof that an attack is happening on a target host, then the end host would end up in the same condition that the protocol is trying to prevent. An extensible scheme should be used so that new authentication schemes can be added at a later date in the case of a flaw being found in existing authentication schemes.

Two conditions must be met with the security of the protocol. First, it must be secure. That is, only the hosts within the group may have the information to authenticate. Second, when the attack discovery is triggered, it must only require a one way communication. That is, no packets can be relied on to be sent back to the attacked host or network. The attacked host or network is to be considered offline during an attack.

Any bit of information in the initiator can be used for authentication. If authentication data for 'Company A' is received, but it requests traceback for a network that isn't owned by 'Company A', the authentication will fail. Only edge routers ever communicate with customer equipment.

Implemented Security

In the testbed, a simple username and password was used. It was sent in plaintext to speed up development of the system. This is, by no means, the final scheme. Since both sides know the password, there is no need to pre-authenticate before the information is sent.

Future Security

In the future, cryptography should be used. Public key cryptography could be used as an extensible authentication scheme. For example, an ISP may have a master key. The routers, while booting, could initiate a public key transfer for all routers on the multicast group. All routers would then send their public keys which were signed by the ISP master key. Then each request would only require being signed by the sending router. The only manual intervention to setup would be the deployment of the ISP master public key. Also, if no known routers or hosts are to be found on a branch of the network, then the router should not listen for initiator packets on the interface attached to that network. If a certain authentication is associated only with a certain network, an initiator packet of another network would be ignored.

Other, non-security related ideas could be implemented. For instance, a token allowance may be issued to clients who pay for the attack discovery service. This would be a revenue generating service for ISPs, as well as prevent abuse of the system.

Routers

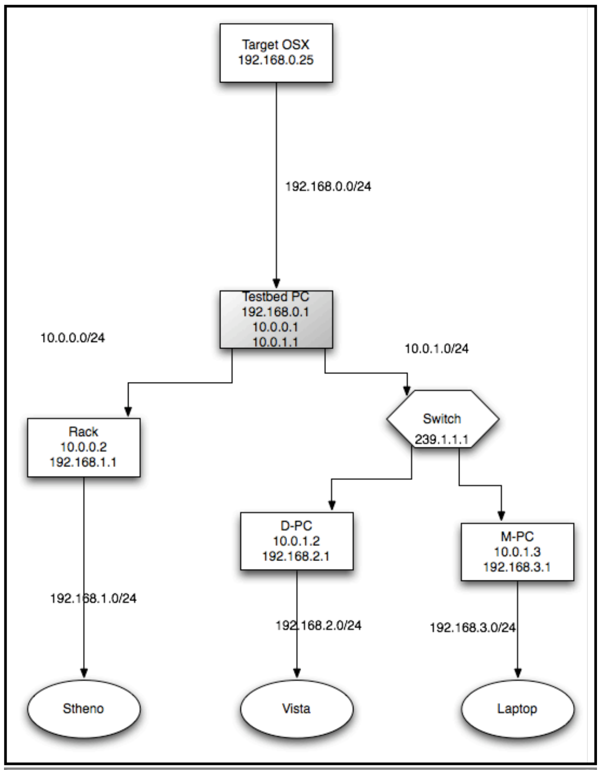
No router access is required between any peers. That is, the ISP requires no access to the customer equipment, nor does the client require any access to the ISP equipment. Only authentication of the actual protocol needs to be granted, in exactly the same way that PPPoE is used on small customer sites to authenticate Internet access. It is also worth reiterating that no client equipment is relied on. Only the closest trusted router handles the requests, and does not pass it on as the client equipment is not considered a peer.

Network Setup

For this research, a simple test bed network was required. It consists of several computers networked together, all running Linux.

Physical Installs

The physical machines that were used vary widely. End nodes consisted primarily of laptops, as they were what was available, and are more than sufficient for testing purposes. The routers were all standard i386 desktops, with multiple network cards. To be thorough, a multi-access network was used between three routers, as well as a point to point ethernet link between one of the three and a fourth router.



Software Installs

The primary OS of the network was Linux. All routers ran Linux, as the daemon was specific to netfilter. The distribution of choice was Debian. The target host, which also included the initiator, was running OSX 10.5 Leopard. Two of the external hosts were running old versions of Gentoo, while the third external host was running Windows Vista. Since this attack is against network protocols, and not software, the specific patch level, and operating systems are unimportant.

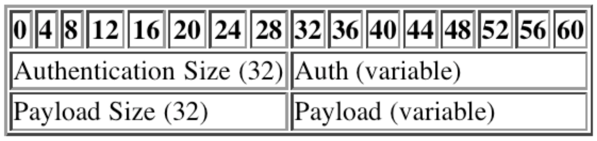
Software Design

The testbed system used requires a basic version of the software to be created. The software is required to have the following functions. It must:

1. Receive packets via multicast.
2. Authenticate the above packets from any source, using an extensible authentication scheme.
3. Monitor for packets matching the signature on all monitored interfaces.
4. Perform an action against packets matching the signature if no upstream peers are present.
5. Generate authenticated packets to pass on the signature to upstream routers, using multicast.

Daemon Development

The daemon was built using C, as it is most native to Linux. The daemon uses UDP port 9494, as an arbitrarily chosen port. The test daemon is also capable of listening to several multicast addresses, one on each interface. In the implementation, unicast, as well as multicast were both supported. This is so that an 'initiator' can communicate with its nearest node without being multicast aware. It also loads a dynamic authentication module for passwords. However, since only one authentication scheme is supported, a true dynamic module loader is not present. Below is a sample packet header as it was used in the testbed system. The authentication section carries authentication data in ASCII. The password authentication format is:
"*password:<username>:<password>*"
where <username> is a username, and <password> is a password. The payload carried a dotted decimal representation of the attacked network, with an optional prefix notation for a subnet. That is: "*192.168.0.0/24*".



0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
Authentication Size (32)								Auth (variable)							
Payload Size (32)								Payload (variable)							

Operating System Integration

Integration with the network is achieved via netfilter. Packets matching the signature, in this case the destination address, add an entry to a recent match, which is accessible via a 'proc' node. Upon a match, the daemon may also create a separate netfilter rule to block, if the interface is setup as such. No other countermeasures were developed in the testbed as blocking provides the simplest method to test the performance of the protocol.

Installation

Since the daemon is configured via a simple configuration file, the only unique installation step to each machine was creating the configuration file. Each router had the daemon installed on it, and then ran as root. It required root access due to the integration with netfilter.

Testing

The system was tested using the aforementioned test network. Network traffic was generated as HTTP downloads. This included many small file downloads, mimicking standard HTTP traffic, as well as large single file downloads. Since only spoof protection was implemented, there were no thresholds based on connection numbers, or amounts of traffic. Statistics were recorded for each of the two types of legitimate traffic before malicious traffic was generated, while malicious traffic was being generated, while the protocol was being triggered, and after the source of the malicious traffic had been discovered.

Normal Traffic

Normal traffic was generated using several web browsers on the network. That is, one on each of the three “user” hosts. While the protocol was being triggered, response times were recorded.

Exceptional Traffic

Exceptional traffic was mimicked by downloading a large file to one or two hosts. This is still legitimate traffic, but is enough to slow the response to other hosts since it is monopolizing network resources. Since this is not considered malicious traffic, the protocol did not stop, or even slow, this traffic. To slow this type of traffic, shaping could be implemented at the web server. In this scenario, it was not implemented. While the protocol was triggered, download speeds were recorded.

Malicious Traffic

Malicious traffic, as defined by the only threshold implemented, is any spoofed traffic. Since the whole point of this protocol is to trace back senders of excess, malicious traffic, the senders were set to flood the web server with traffic. The traffic was not legitimate, and did not TCP Checksum correctly. This meant that the web server merely dropped the traffic before it could reach the web server daemon, and ruled out TCP SYN flood from the attack.

Results

The first phase of testing was to establish baselines of the network. To simulate a proper network connection, access lines were limited to 10 Mbps, half duplex, where network cards allowed. Most importantly, this included the web servers connection. Response times for standard web traffic from all three “user” hosts were under a second. Ping times were less than 1 ms. Exceptional traffic was clocked at 900KB/s, divided between each connecting host. That is, if only one was connected, it experienced 900KB/s. If two were connected, each would experience 450KB/s. On a 10 Mbps connection, this is the most that can be expected, which is around 7.2 Mbps. Immediately after a single host started packet flooding, response times jumped to 30 seconds or more. Packet loss approached 100%. Exceptional traffic speed plummeted to bursting between 0KB/s and 3KB/s.

Response times without the protection algorithm

The first step in responding to an attack is detecting an attack. Some factors can decrease this time, such as a network monitoring solution or IDS. However, smaller networks may only be alerted when users are unable to visit web pages or receive email. I will label the time to detect an attack as *D*. One important note is that few network administrators have any type of access to their ISP’s routers. That is, to do any kind of tracing or mitigation, they would need the cooperation of the ISP. For requesting a response from the ISP, I will label this *R.* Once the ISP has been notified, their support technicians must be contacted, as few, if any, ISPs have their technical staff answering support calls. I will call this notification time *N*. Once a technical person has been informed, they must then start the process of discovering the source. Each router along the chain would need to be connected to, checked, and possibly have a packet filter enabled. I will denote this process of checking each router as *C*. Since more than 1 router needs to be contacted, we would have to multiply out by each router that requires contacting. I will denote the number of routers that need to be contacted with *n*. This leaves us with the total time *T*.

T = D + R + N + (C x n)

If we take a small company, without an IDS or network monitor, we can assign D as 10 minutes. I will assume a diligent ISP with sufficient phone support staff to promptly handle the call, and assign R as 1 minute, and N as 1 minute as well. With a fast typing technical staff, we can hope for 30 seconds per router (C). If we guess at 4 routers (n), all accessible to the ISP technical staff member, we are left with:

T = 10 + 1 + 1 + (0.5 x 4)
T = 14 minute

If the company has an IDS or other network monitoring, we could assume 1 minute for D, leaving us with 5 minutes response.

Unfortunately, the Internet is not quite that simple. With many ISPs, and none giving access to their neighbors, we need to multiply the number of ASs. We can multiply all of the numbers by the number of ASs (*A*) by everything but the detection time.

T = D + (A x (R + N + (C x n)))

In this extended case, R, N, C, and n would all be average values of their singular counterparts between the ASs. If we use the numbers from the first example, but between 2 ASs, we get:

T = 10 + (2 x (1 + 1 + (0.5 x 4)))
T = 18 minutes

We can very quickly see how fast these numbers may climb as more routers are added. Deep call queues of an ISP could translate to even slower times.

Response times with the protection algorithm

Once the initiator was triggered, response times for regular traffic was restored to its normal state. Exceptional traffic was again clocked at 900KB/s. However, the most important mark of the performance of the protocol is the response time. Once triggered, normal speeds were restored in between one and two seconds. This speed is due to the 1 second timer implemented in the daemon’s code in order to ensure netfilter was finished adding the rules. If netfilter was triggered too quickly, it would merely report that the resource was not ready, and the rule would not be added. With a more closely integrated daemon, this time could be decreased drastically.

Shortcomings

The first shortcoming with the proposed solution arises when large portions of the Internet don’t participate. When a router has no peers out of its interface, the default policy is to block packets destined for the attacked network. So if a section of the Internet, such as Asia, does not participate, then an attack from Asia would prevent legitimate traffic from accessing the destination. This could be overcome by adding an options field to the initiator packet that gives the attacked host the ability to inform all routers that it is OK to block certain non-conforming areas of the Internet. That is, a local news website in Florida may not care if all of Asia gets blocked, as their content is not relevant to Asia anyways. Or a retailer deems that it is OK to block parts of Africa, as they don't ship their products outside of North America. Instead of blocking, another action could be implemented, such as was mentioned earlier. An IDS could be used to filter malicious traffic from a non-participating AS, instead of blindly blocking traffic. Since only traffic destined for the attacked host would need to be processed, it could be a viable option over blindly deploying a filtering system for the entire AS ingress. Ultimately, blocking traffic from an external AS to the attacked network is much more desirable than having all traffic from all networks being dropped. That is, a partial DDoS is favorable over a complete DDoS. In the end, it is the attacked host that controls the criteria for detection.

Other protocols support forensic analysis of packets from the past to determine the attacker's location. This protocol is only effective against stopping current attacks and does not serve to reconstruct the attack path after the attack has finished. As I have mentioned previously, protocols that depend on the attacked host require a second augmentation to mitigate a current attack. They also depend on the attackers packets reaching the host to be processed, which may not be possible.

Conclusions

DDoS attacks are common on the Internet. There are current solutions, but they require the target for the attack to have a large budget to combat the problem. Since response times from manual intervention are slow, a faster method is needed to combat the problem. This method is one such approach. Its simplicity does not demand huge resources to be spent implementing it, and its design makes it work in an environment that doesn't require time consuming patches to be applied, nor maintained. Since attack response can be triggered by clients, ISPs can easily implement accounting controls as a revenue generator to pay for the implementation of the protocol.

References

[1] Richardson, R, CSI Computer Crime and Security Survey 2007

[2] C. Douligeris and A. Mitrokotsa, "DDoS Attacks and Defense Mechanisms: Classification and State-of-the-Art," Computer Net-
works, vol. 44, no. 5, pp. 643-666, Apr. 2004.

[3] Cisco Systems, Cisco Distributed Denial of Service Protection
Solution: Leading DDoS Protection for Service Providers and their Customers (http://www.cisco.com/cdc_content_elements/networking_solutions/service_provider/ddos_protection_sol/ddos_protection_bdm_wp_0602.pdf)

[4] Chen, Ling, Chow, Xia, (2007). AID: A global anti-DoS service, Computer Networks
(Vol/Issue: 51 (15), Date: Oct 24, 2007, Page: 4252)

[5] Tian Bu, Samphel Norden, Thomas Woo, A survivable DoS-resistant overlay network, Computer Networks 50 (2006) 1281–1301

[6] Gu Hsin Lai, Chia-Mei Chen, Bing-Chiang Jeng and Willams Chao, Ant-based IP traceback, Expert Systems with ApplicationsVolume 34, Issue 4, , May 2008, Pages 3071-3080. (http://www.sciencedirect.com/science/article/B6V03-4P40KHS-1/2/69a48379b79a6f49ae5df419813515f6)

[7] Bhaskaran, V. Murali, Natarajan, A. M., Sivanandam, S. N., A New Promising IP Traceback Approach and its Comparison with Existing Approaches., Information Technology Journal (2007, Vol. 6 Issue 2, p182-188, 7p)

[8] RFC 2827

[9] Chen, R, Park, J, Marchany, R, A Divide-and-Conquer Strategy for Thwarting Distributed Denial-of-Service Attacks, IEEE Transactions on Parallel and Distributed Systems (VOL. 18, NO. 5, May 2007, p577-588)

[10] Yu Chen, Kai Hwang, Wei -Shinn Ku, Collaborative Detection of DDoS Attacks over Multiple Network Domains, IEEE Transactions on Parallel and Distributed Systems, VOL. 18, NO. 12, December 2007

[11] Cisco Internet Protocol (IP) Multicast (http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ipmulti.htm)