Towards A Secure Network: Defense vs. Attack[1]

H M N Dilum Bandara, Yan Chen and Douglas Tear
Colorado State University, Fort Collins, CO 80523, USA.

Abstract — Security threats are becoming more prevalent in the current Internet. The objective of the assignment is to design and build an effective defense and attack strategy and monitor network performance and robustness under variety of attacks. A test bed of 3 small autonomous systems running standard services such as DNS, web and e-mail was setup. Automated clients scripts were developed to simulate traffic in the network. A 4th network was set up to act as an arbitrator and measured performance during attacks.

As team2, we designed the network with security in mind. MRTG, Snort and ntop were implemented to monitor the network traffic and suspected attacks. In terms of security, a multi-layered protection scheme is used to protect all services. Each service is implemented in a different subnet to easily apply security policies based on services and to separate traffic. Network access policies were implemented at the router using Access Lists and IPTables was used as the firewall. Application specific security features were configured to make the services robust. All the events were loged We describe our defense and attacks strategy and the effectiveness of those approaches. Router became the bottleneck while defending due to lower processing ability. The Shrew attack on team1 was highly successful without degrading the performance of our services.

***Index terms* — Attacks, Defense, Network Security**

A. INTRODUCTION

Due to the open nature, the current Internet is suffering from a variety of threats such as worms, viruses, spyware, DOS and DDoS attacks, spam, etc. The objective of the exercise was to design and build an effective defense and attack strategy and monitor the network performance under a variety of attacks while gaining hands on experience in various aspects of networking and security. The exercise requirement was to construct a small network that reliably provides common services such as web, e-mail, and DNS. Several automated clients were needed to keep the traffic flowing in the network while utilizing those services.

Security is the main concern in our network. The network should be secure enough to protect valuable services, sensitive data, and network resources from unauthorized access. At the same time, it should be robust enough to provide services to users under severe attacks. In addition to defending our network, we needed to develop some tools to attack other networks by leveraging the vulnerability of those networks.

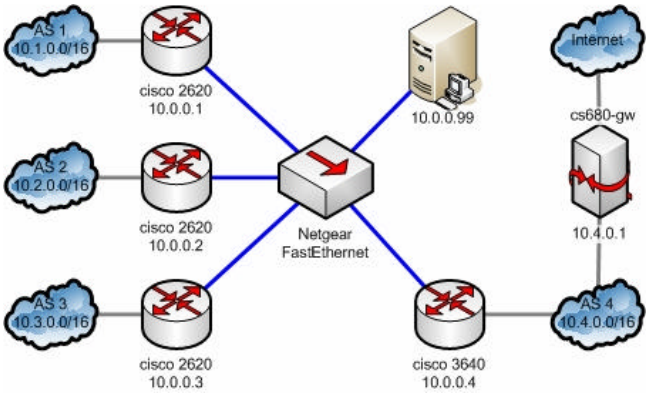

Fig. 1. WAN network topology

As Fig. 1 shows, there are 3 teams. Each team's network is considered as an Autonomous System (AS) and assigned a unique AS number and IP address block. Besides these 3 ASes, another AS was set up as a gateway to provide access from the Internet so that each team member can remotely access their network. The border routers of all ASes are connected via a Fast Ethernet switch. Each AS is peering with the other 3 ASes through BGP sessions to exchange their routes. Additionally, a root DNS server was setup on a separate host which is directly connected to the switch. This also acts as the arbitrator.

As team 2, we were assigned an IP address block of *10.2.0.0/16* and use the domain name *as2.lab*. To setup our AS we used 6 Pentium III PCs, a Cisco 2620 router and Cisco Catalyst 1900 switch. All the hosts were running unpatched version of Fedora Core 6. Apache was selected as the web server and the mail transfer

[1] This is a term paper for CS680: Advanced Topics in Networking - Network Security, Spring 2007.

agent was Postfix. DNS servers were based on BIND. The Multi Router Traffic Grapher (MRTG) [3] is used to monitor both incoming and outgoing traffic at the border router. Ntop [2] is used to measure the traffic in 5 second intervals. Wireshark [7] is used as a network protocol analyzer to troubleshoot network problems by sniffing and analyzing the traffic in the network. TCPdump [8] is used as a basic traffic sniffer. SNORT [4] is used to detect potential attacks on our network and Nessus [6] is used to scan the network for vulnerabilities. IPTables [9] is used as a firewall to further protect services.

The rest of the paper is organized as follows. Section B describes the network architecture we designed and implemented. Section C describes the services and baseline. Section D describes the defense strategy and its implementation. Section E provides observations under attacks. Attack strategy and effectiveness is described in section F. Section G further discusses the network architecture, attacks and defense and it is followed by concluding remarks.

B. Network Architecture

The main goal was to establish a network that is easier to manage. The layout of the network, services and IP addressees are illustrated in Fig. 2. The network was connected to the other two ASes through the 100Mbps gateway switch. The Fast Ethernet port in the router was connected to the gateway switch in order to gain higher bandwidth. Since we did not have crossover cables the remaining 4 Ethernet ports were connect to individual hosts through the VLAN enabled switch. A separate VLAN was configured for each router-host connection and this later allowed us to put more control on traffic flow. For an example Eth 1/1 was connected to the mail server and all the incoming SMTP requests were forwarded only to this port. This also allows higher bandwidth from router to hosts.

The router acted as the DHCP server for Host1-4 and IP addresses were given based on hosts MAC address. Each LAN has a /24 address space. Host1 was running the DNS server, FTP server and TFTP server. Mail server was running on Host2 and Host4 was running the web server. Host4 was connected through Host3 which acts as a firewall and it also hosts the secondary DNS server. We used Host5 predominantly for attacking and since it was directly connected to the router it had a 10Mbps outgoing bandwidth. Snort and ntop were running on Host6. All the traffic through the switch was mirrored to this host.

C. Services

The DNS was the first service to be configured. Bind 9.3.2 was used as a domain name service and was resolving names for hosts and services for as2.lab domain and was able to resolve names on other ASs through the root name servers configured in AS4. It was also configured to perform reverse name lookups. Each entry had a timeout of 10 seconds. This was mostly done for the purpose of increasing traffic in the network. The DNS was configured to log all the queries but later this was disable due to increased log size. For the purpose of fault tolerance a secondary name server was configured on Host3. It transferred the zone information from the

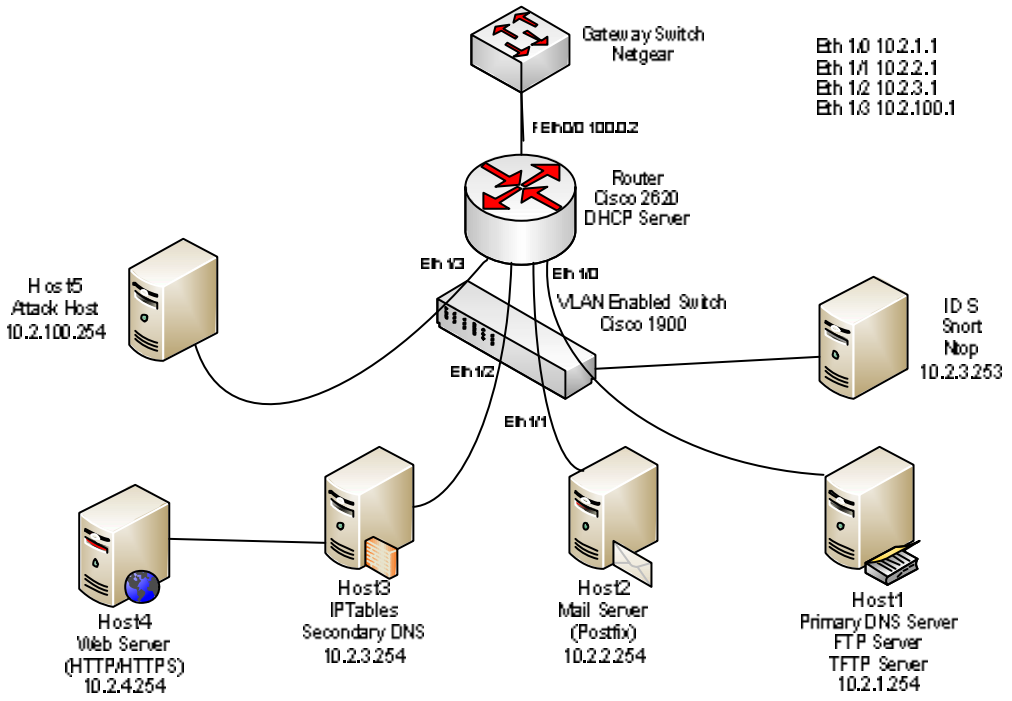

Fig. 2. Network layout

primary server and an entry was also added to the root name server.

SSH was enabled on all the hosts. Host1 also hosts the TFTP server for the purpose of uploading router image and to backup router configuration. File Server was also configured using the very secure FTP service.

Postfix 2.3.3-2 is used as the mail server. Postfix became the first choice since it is easier to configure and it is renowned for higher security. It was setup with the objectives of later adding spam and virus filtering however those features were never implemented. It was configured to accept e-mails for the *as2.lab* domain and users were able to check mail using any IMAP client. Courier-imap was used to provide IMAP access to Maildir mailboxes. Since it was a bit harder to work with IMAP clients SquirrelMail was installed as a WebMail service. Six user accounts were added, 3 for the group members and 3 dummy accounts, namely user1-3. It was seen that the mail server was able to receive more than 90 mails per second. Since the mail server was generating a large log file logs were rotated in every 3 hours.

After all ASs agreed on the 10 request per second based line it was seen that the arbitrator was able to send 6.5 e-mails per second in average (table 1) and the server was receiving an aggregated average of 16.7 mails per second.

Table 1 – Baseline performances

Services	Arbitrator requests	Aggregated requests
Mail	6.5	17.6
Web	8.0	31.1

Web server based on Apache 2.2.3 was configured on Host4. It hosts the AS2 web sites which had 100 web pages with a mean of 10000 bytes. We also had a secure version of the website under *https://www.as2.lab*. MRTG graphs were also visible through the web server. Under baseline operations the arbitrator was able to receive 8 web pages per second in average (Table 1) and server was catering an aggregated average of 31.1 requests per second.

Although the other two ASs were replicating their services to increase fault tolerance we decided not to do so because of three reasons. We needed to build a simple system as done by most of the small size organizations; secondly we want to see how effective our approach in withstanding an attack; we also realized machines were becoming a bottleneck in handling large requests so we decided not to run multiple services on the same host.

A. Clients

Both mail and web clients were configured to send the baseline of 10 requests/second for each AS in average. Both clients were written in Perl script.

Mail clients initially used Sendmail to send the mails and those mails were relayed through the mail server. However there was an error, after sending mails for a couple of hours Sendmail suddenly stop forwarding any mails and there were multiple corrupted instances of Sendmail in the process table. Then we switched to the scheme of sending mails directly to the receiver's mail server using some SMTP libraries in Perl. A total of 10 mails were send approximately every second depending on the delay in the establishing and delivering the mail. Three clients were running on Host1-3 and 2 of them send 4 mails/second/AS and the other one sends 2 mails/second/AS. The number of connection attempts and successful attempts for a given time period was logged.

The web clients use a simple wget call to each AS. The wget was configured to try once for each attempt and records a success when successful. A total of 10 requests were made approximately every second across the clients depending on the delay in the wget function. The clients were distributed across Host1-4. This was done to balance the load across the network and to avoid overloading a single machine. Each client on hosts 2 and 4 made 2 requests/second while clients on hosts 1 and 3 made 3 requests per second. After 60 rounds of requests the total number of successful attempts were average out and the results were saved to a log file.

Two design errors in the script were determined after the attack scenarios. The first was the timeout value of wget was not changed from the default value. This caused wget to hang while it was waiting for a reply from the server host. This resulted in the success rate not being calculated approximately every minute, but the duration lasted up to several minutes during the attack. The results were aggregated across several minutes and did not give a detailed account of the performance of the network being monitored during the attack. The second error was iterating the outer loop 60 times to approximate a one minute time frame. Even though the success rate was calculated on actual time, it was not calculated at equal time intervals during the iteration. The solution was to use a timer to calculate the success rate at a specific interval. This would also allow the time interval to be changed to a longer or shorter interval than 1 minute.

D. DEFENSE STRATEGY

We use the defense in depth approach while designing the defense strategy. We configured multiple levels of defense at the router, IPTables and at the application level. Following sections describe the defense strategy in detail. We used both a proactive and reactive defense strategy.

A. Defense At The Network Level

We wanted to ensure that network and application will not become the vulnerable point during an attack. Since we were not supposed to patch any of the hosts and applications we put our best effort to put maximum defense at the network level. Basically, the router is used to provide general security protection. And IPTables is used to protect more important services.

Routers

We first protected the router itself and then configured Access Control Lists (ACLs) for other services.

Since more services open more vulnerabilities, our first security policy was to disable any unnecessary service on the router. For example, IP redirect is not useful in our network environment. However, this function is turned on in the router. It can amplify SMURF or FRAGGLE attacks or used to set up Man-in-the-Middle attacks. Turning off such functions removes potential vulnerabilities without affecting legitimate traffic.

Additionally, the router used in our network does not support SSH. Since the communication using Telnet is not encrypted, our router was set up to only accept remote login from the specific machine in our network.

Routers mainly operate on IP layer and are used to route the packets according their destinations. Lots of security features are implemented in routers. One of the most important features is the use of ACLs. The basic function of ACLs is to look at source and possibly destination IP addresses to make sure that only packets from/to authorized users are allowed to go through. Furthermore, an extended ACL can filter the packets based on the network protocol used, TCP/UDP source or destination port number, or ICMP types. Take Cisco router as an instance. Following configuration shows that the router only allows web traffic from 10.1.0.0/16 to host 10.2.4.254.

```
access-list 110 permit tcp 10.1.0.0 0.0.255.255
host 10.2.4.254 eq www
Deny any any
```

Another security feature is route-map, which is similar to access list. It is used when the route, that a packet takes needs to be altered. Normally, it uses ACLs to define the conditions. When the conditions are met, an action can be taken. Actions are defined using set commands and can be used to modify the packet or routes. In this exercise, we use route-map to filter unwanted routes learned from BGP to prevent prefix hijacks. Normally, prefix hijacks can be easily achieved by announcing a more specific route. For example, in our exercise environment, AS3 can hijack our traffic from to AS1 by advertising 10.1.0.0/17 and 10.1.128.0/17 to our router. Since these two routes are more specific than the route we learn from AS1, by longest match rule, our router will pick the routes learned from AS3. In this case, AS3 successfully hijacks all our traffic to AS1. To prevent the above hijack, we set up following rule using route-map.

```
A route announced by peer p will be accepted if
and only if the ip space contained in the route
falls into the ip space owned the by p.
```

Note that under above rule, if the link or the BGP session between AS2 and AS1 is broken, the traffic from AS2 to AS1 will be lost and vice verse. The reason is that AS2 will never believe the route about AS1's IP space announced by AS3.

Besides filter routes learned from BGP, we use ACLs to control the incoming traffic and the traffic to each subnet. Since we separate each service into different subnets, we apply ACLs on the interfaces connecting the service server. For example, only DNS traffic is forwarded out to the interface connecting DNS server's subnet. In this way, even if one of the servers is compromised, it is still hard for attacker to use that server to attack the other legitimate servers.

Nowadays, lots of attacks are based on spoofing the source IP address in the packets. In this way, attackers can either make themselves hard to be detected or easily launch a DoS attack.

To mitigate the problem caused by forged IP addresses, Cisco provides a nice feature called Unicast Reverse Path Forwarding (RPF) [10]. Basically, Unicast RPF is applied on an interface. The router examines all packets received as input on that interface to make sure that the source address and source interface appear in the routing table and match the interface on which the packet was received. If they do not match, the packets will be dropped. Unicast RPF is useful when the path between source and destination is in symmetry. In our network topology, although each BGP router connects to the other 3 routers by one interface, we can still remove the forged prefixes not in the allocated IP space.

Table 2 – Summary of IPTable rules

Host	Rules
Common rules	SSH only with AS2 network and gateway machine Outgoing DNS, ICMP ECHO, DHCP requests One ICMP ECHO replies per second
Host1	Incoming DNS requests only Outgoing mail and web for clients 20 TCP SYN per second
Host2	Incoming mails only Outgoing mail and web request for clients 50 TCP SYN per second
Host3	Incoming DNS requests only, zone transfer only from primary server Outgoing mail and web for clients Forwarding only web traffic to Host4 20 TCP SYN per second
Host4	Incoming web requests only Outgoing web for clients 50 TCP SYN per second

IPTables

Before configuring any rules a network scan was performed on all the hosts in the network using Nessus vulnerability scanner [6]. This network scan reviled some of the services that were running without our knowledge. Some of those include; sunrpc running on TCP/UDP port 111, FTP and TFTP that we forgot to disable, unknown service running on TCP/UDP port 814, mdns on UDP port 5353, POP3S on TCP port 995, swat on TCP 901, etc. Most of the services that we could disable were stopped and for other services specific REJECT rules were added to IPTables. Default policy was to block everything. Since we did not require running FTP and TFTP during attacks those services were disabled. Host3 added additional protection to web server by having another security layer. Only web requests were forwarded to the web server. Table 2 summarizes IPTable rules for each host. After applying IPTable rules and enabling SeLinux on Host1-4 another vulnerability scan was performed to make sure that the defenses were in place.

B. Defense At The Application Level

Although we were not able to patch any of the services we were able to put some constrains at the application level by using some of the built in features in individual services.

In order to ensure fault tolerance a secondary DNS server was installed. This DNS was configured not to accept any zone transfers other than from the primary DNS server.

The mail server was configured to relay mails only within AS2 IP addresses. Mails that were sent to unused accounts were immediately rejected to prevent any resource utilization. The server was configured to access only 50 concurrent mail connections (although we saw it can handle more than 90 connections) and parallel delivery limit to same destination was set to 10. These were done to prevent the server from being overloaded during an attack. In order to prevent the mailboxes from filling the hard disk, mails were deleted in every 60 seconds and deferred queue was flushed every 3 hours. Mail log was also rotated in every 3 hours.

The Apache web server has some built in features that we used to try and make or web server more robust to an attack. A 30 second timeout directive on send and receive connections was set. This directive determines that a connection will time out after 30 seconds due to one of three reasons; a late GET request, time between TCP packets on a POST or PUT request exceeded, or time between ACKs on TCP packet responses exceeded. This prevents an unresponsive or latent connection from being kept open and occupying idle server processes.

The MaxKeepAliveRequests directive was set to a value of 50. This only allowed 50 requests on a persistent connection. This prevents large number of persistent connections from dominating all the connections to the server by continually making a request and using up server resources. Although in the real world this would inhibit performance to some clients, but it allows a greater number of clients a share of the server's resources.

Finally MaxAliveTimeout directive was set. This directive will timeout a persistent connection if a new request is not made within the given timeout value. The value was set to 15 seconds to prevent an inactive connection from occupying the server resources.

These different rate limits were set in an attempt to prevent the resources of the mail and web server from being dominated under an attack by malicious clients resulting in a denial of service to legitimate clients.

C. Network Monitoring

In a defense scheme, detecting anomaly in time is a prerequisite condition to reduce the effect of an attack. Furthermore, the monitoring data is also useful for us to analyze the behavior of attacks and track back to the attackers.

We planned to defend our network during an attack using several approaches. Before we put any dynamic controls on the network or applications we need to

understand what is going wrong in the network. With that objective we install two traffic analysis tools MRTG and ntop. Initially we had MRTG installed and it monitors SNMP statistics from SNMP-capable devices. In this exercise, we monitor our incoming and outgoing traffic and provide traffic breakdown by application layer protocols. Additionally, we use MRTG to check each server's resource load, such as CPU utilization, memory utilization, and the number of TCP connections, etc. The drawback of MRTG is that it cannot reflect traffic change in time since it collects data every 5 minutes. Because of this delay we used ntop. The ntop provides more instantaneous response of network traffic, therefore ntop was preferred over MRTG. The drawback of this approach is that we cannot directly distinguish incoming traffic from outgoing traffic.

We also installed an Intruder Detection System (IDS) to further support our attack detection. Snort [4] was selected since it was much easier to configure and use than some of the other similar products. All traffic through the switch was mirrored into Host6 to be monitored by Snort. Snort was configured in *alert mode* using a basic set of rules provided by the Snort website. All alerts generated by snort were then sent to a log to be monitored during the attack.

Some of the rules included in the basic configuration were: DoS, DDoS, SMTP, Telnet, ICMP, SNMP, attack-response, web-client, FTP, and other-IDS rules. Rules were picked trying to anticipate the basic possible attacks on our network. Snort was able to detect most of the attacks during the vulnerability scan on our own network.

D. Plans For Defending While Under Attack

We were connected to all the hosts and router through the terminal and remotely so that we can easily manage any issues on the machines, services and router. We were depending on Snort, ntop, mail and web logs to give us some hints while we were under attacked. We were planning to put rate limits at the router based on attacks that we detected. Similarly we also had some IPTable template rules to quickly block or rate limit detected attacks. We also identified some parameters in the web and the mail servers that we could change in order to reduce the effect of the attack. Examples include blocking email having a specific header or body, rate limiting concurrent connections and changing timeouts.

E. OBSERVATIONS UNDER ATTACK

We were attacked by AS3. During the attack, our router stopped responding to any requests from our terminal that connects to the router through the console port. However, our monitor did not detect any anomalous traffic. Our conjecture is that attackers were sending lots of packets, such as TCP SYN packet and Telnet requests to the router. Those packets were blocked by our security policy. In order to check if the incoming packets match a given rule or not, the router has to look into the TCP header in each and every packet. These operations took too much CPU utilization and eventually reduced packet forwarding to a very low value.

Since we did not set up our monitor before the router, we cannot verify our conjecture. After the presentations, the attackers verified our conjecture.

Table 3 – Measurements by arbitrator (success/second)

Service	AS	Baseline	Attack		
			AS3→AS2	AS2→AS1	AS1→AS3
e-mail	1	6.4	6.8	1.8	1.7
	2	6.5	2.3	6.5	4.8
	3	7.2	4.2	7.3	6.4
Web	1	8.6	8.7	1.8	3.9
	2	8	2.7	8.1	6.8
	3	8.8	5.7	8.8	2.3

Unfortunately, Snort did not generate any useful alerts during the attack. This could have been due to a couple of reasons. Not enough traffic was able to pass through the router into the switch during the attack. In this case, there were not enough attack packets reaching the Snort machine to signal an alert. Another reason could have been the snort rules were not configured for the types of attacks made on our network.

There was a drop in traffic indicated by ntop. Both web and mail log did not reveal any suspicious behavior. Later it was reviled that attackers did perform a form of Shrew [1] attack on our mail and web servers however the attack die after the first 2 minutes. This is clearly visible in figure 3.

A. Performance under attack

During the attack we observed a significant decrease in services. The router was our first line of defense against an attack, but turned out to be our downfall. This became evident during the attack as the router became unresponsive to Telnet and management

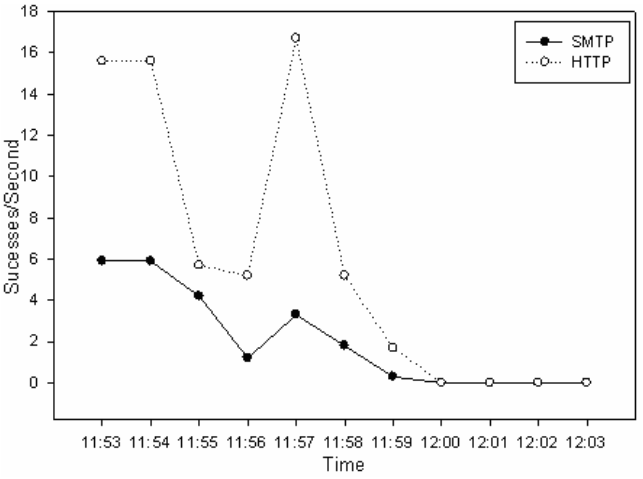

Fig. 3. AS3 attack on AS2 as observed by AS1

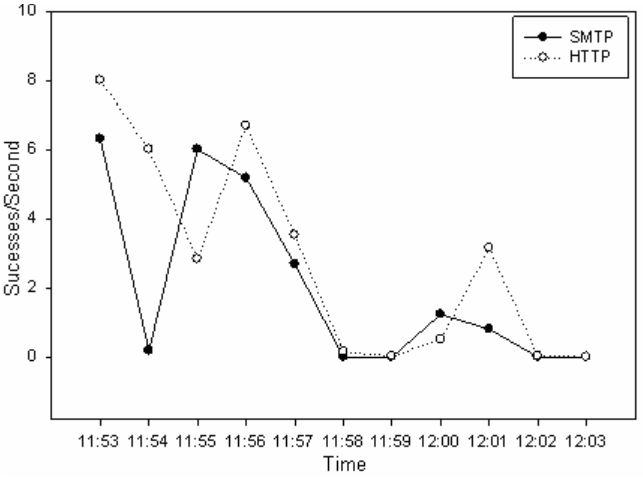

Fig. 4. AS3 attack on AS2 as observed from log files

console. During the attack we attempted to log into the router and set rate limits on the traffic originating from AS3, but were unable to connect to the router. The router was too busy analyzing and applying security rules rather than forwarding traffic. Since the router was fairly old it did not have much processing power therefore all its resources were utilized much faster.

The affects of AS3's attack can be seen in Table 3 which shows the measurements recorded by the arbitrator. The 3rd column is a baseline average recorded over a 10 minute interval before any attacks were launched and traffic had a chance to stabilize. The next 3 columns represent the recorded averages over a 10 minute time period during each of the 3 attacks:

Attack 1 – AS3 attacked AS2

Attack 2 – AS2 attacked AS1

Attack 3 – AS1 attacked AS3

All figures represent the average successful requests per second at the end of the 10 minute time frame as averaged over the entire time frame.

As can be seen in Table 3, the 10 minute average for web requests dropped from the baseline of 8 successful requests per second to 2.7 requests per second. The average for sent emails dropped from the baseline of 6.5 successful transmissions per second to 2.3 successful transmissions per second. However like some of the other ASs our service rate never went to 0. This data is a bit misleading since the arbitrator has an accumulative average.

As the attack progressed the performance of the network degraded significantly. Fig. 3 shows the results from the clients in AS1 which continued to try and send requests to AS2 during the attack.

As can be seen, the performance of the servers degraded to 0 during the last few minutes of the attack. The graph shows a curious peak during the attack where the network appears to almost return to normal. The only explanation we can offer is AS3 tried to launch a type of Shrew attack. Shortly into the attack period AS3 reported their Shrew attack failed and stopped running. We surmise this could have led to the spike in network performance before AS3's other attacks were able to fill the void and shutdown our network.

Upon examination of our web and mail server logs, there were some successful requests made during the last minutes of the attack. This shows a limited number of requests were able to pass through the router to the web and mail server. However, these successful requests originated mainly from the arbitrator at AS4 and a few request originated from AS3, but none of the requests originated from AS1 as reflected in their data and this was also visible in our logs. This accounts for the discrepancies between the arbitrator and AS1. The arbitrator was able to make some successful requests during the attack, but AS1 was not. We do not have an answer for these discrepancy.

Fig. 4 shows the successful requests from our server logs based on requests made by the arbitrator. Table 3 shows that during AS3's attack on AS2, the arbitrator's success rate to AS3 dropped significantly. The success rate for email dropped from the baseline of 7.2 per second to 4.2 per second and the web success rate dropped from 8.8 per second to 5.7 per second. We believe this suggests that AS3's attack was having a negative effect on their own network and degrading the performance of their own services.

An interesting note, the Apache log shows AS1 was initiating approximately 16.6 requests per second versus 8.9 average requests per second from AS3. AS1's requests were almost twice the number of requests from AS3 and much greater than the specified approximate 10 requests per second. It appears AS1 was running 2 hosts

each making approximately 10 requests per second instead of the 10 total requests.

None of the requests from AS1 were answered successfully in the last minutes of the attack unlike some of the successful requests of AS3 and AS4. We were not able to derive a conclusive reason for this behavior.

F. Attack Strategy

We crafted an attack that consists of a Shrew attack and TCP SYN and Reset attack.

A. Shrew Attack

We attempted to implement a shrew [1] attack on AS1. Shrew is an attack which exploits TCP's retransmission timeout mechanism. The shrew attempts to congest a network with a burst of packets long enough to cause a TCP connection to enter timeout. The shrew attack will then stop transmitting for the timeout period before sending another congestion causing burst of packets. By sending the attack in bursts, the shrew minimizes the total throughput of the attack. Yet it still is able to cause a disruption in the TCP connections without a large flood of attack packets, allowing it to minimize its chance of detection.

In our shrew attack, we used PackIt [5] to craft attack packets of maximum size in order to try and cause congestion in the network being attacked. We attempted to create a burst of packets of length 0.1 seconds as suggested in [1]. Several trials were performed with PackIt to see how many packets of maximum size could be sent in one second and then this number was divided by 10 to try and create a 0.1 second burst of packets. It was determined over the trials that 9000 – 9100 packets were sent per second at full rate. A burst size of 91 packets was decided upon to create an approximate 0.1 second burst of packets.

Next a timeout between burst was set at 1 second following the results given in [1]. This timeout resulted in an approximate 15 - 20 percent decrease in successful responses using the arbitrator on Host4 as a measure. In order to try and achieve an increase in the affect of the attack, the timeout period was cut in half to 0.5 seconds which resulted in similar results in [1]. This appeared to increase the effectiveness of the attack in the few trails that were performed in the limited time we had left.

To increase the effectiveness of our attack, 2 types of packets were used in the attack in order to try and maximize the congestion in the network being attacked. This was also done in an attempt to avoid any defensive measures by the network being attacked by choosing common packet types for services implemented in the attacked network. The first set of attack packets were TCP SYN packets sent to TCP port 80 (HTTP), and the second set of attack packets were UDP packets sent to UDP port 53 (DNS).

Since the other two ASes had replicated their services across all of their hosts, both types of packets were sent to each of their hosts. To try and further limit the possibility of detection and any rate limiting that may have been applied by the AS, each attack packet flow had a different crafted source IP address, either from AS2 or AS3.

In the few trials that were able to be preformed, the attack seemed to have a significant affect on AS1, but it is unknown if this was actually the intended result of the attack or the result of the router not being able to handle the burst of packets. Further trials and measures would be needed.

B. TCP SYN Reset On Router

A simple TCP SYN and Reset attack was done to AS1's router. The attack was directed towards the BGP port (TCP 179) of the router. A total of 10000 attack packets were sent at a burst of 10 packets. The source IP was spoofed to one of the AS1's mail servers since we knew it was slow in handing mails. So we wanted to make it even slower by asking it to handle packets that it didn't send. Following command was used to generate the attack packets using PackIt.

```
packit -s 10.1.0.5 -d 10.1.0.1 -S 403 -D 179 -F SR -q 12345678910 -c 10000 -b 10
```

We do now have an idea of how successful the attack was. This was not a well prepared attack it was just written on the fly because we were not sure about the effectiveness of Shrew attack. However we assume that least it kept the victim router busy.

C. Performance under attack

Our attack on AS3 resulted in a significant decrease in the performance of their services. The 3rd column of Table 3 shows the results of the successful request from the arbitrator on AS4 during our attack on AS1. As can be seen, after the 10 minute attack period, their average successful email rate dropped from 6.4 per second to 1.8 per second. The average successful web request dropped from 8.6 per second to 1.8 per second. Once again this is an average over the 10 minute attack period and does not truly show the total effects of the attack.

During the attack, AS1 was forced to unplug their router from the network in order to log into it, in an attempt to thwart the attack. It appears our attack had the same effect as AS3's attack had on us. Their router

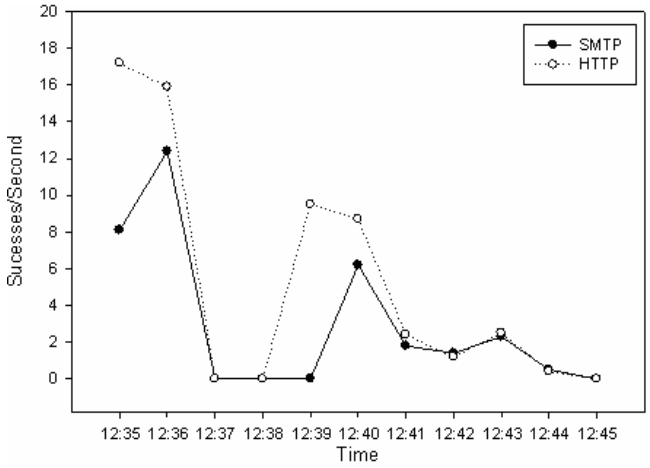

Fig. 5. AS2 attack on AS1 as observed by AS3

was not able to process the number of packets being sent to them.

Fig. 5 shows the client logs from AS3 as they were monitoring the successful requests from AS1 and AS2. AS3 monitored both ASes together so the results are aggregated between the 2 ASes. AS3 had a few time points in their data with anomalous readings. The flat spot in the middle of the graph had no useable data points. These 2 factors made it hard to determine if the decline in services was due to AS1's lack of response because of the attack from AS2 or factors with AS2. However, a decline in services is shown.

AS2 was the only AS not to show degradation from baseline while we launched our attack. AS1 and AS3 both showed a decline in successful requests from the arbitrator during the launch of their own attacks. This is because we only used 1 attack host and it was directly connected to the router.

It was disappointing not to be able to see the actual effects of out attack. We were very anxious to see if and how effective our attacks would have been.

G. DISCUSSION

The router was our first line of defense against an attack, but turned out to be our downfall. In hindsight there were two misjudgments made on our part. The first of which, the Snort box should have been placed in front of the router to monitor and filter all incoming traffic. This would have made it easier to detect and stop a subsequent attack. Next, Snort could have been used to monitor the packets coming into the network to manually identify possible attacks. However, this might have been practically impossible due to the amount of packets being flooded into the network while it was under attack.

If Snort was configured in the filter mode, it might drop enough attack packets so that the router would not get overloaded.

Another solution would be to place a firewall in front of the router. However this is not done in practice. Although this can be done in an enterprise network it would be impossible to place such a firewall in the backbone network.

Even if we place firewalls and monitoring tools to detect what is happening, if the attacker targeted these, it still may not be effective. A more practical solution would be to block such attack packets at the source network.

It was hard to derive any concrete conclusions from the data collected by each AS because they were so different and we also saw that some hosts were not time synchronized. Although we expected that the arbitrator would give a clear picture of success rates it actually gave accumulative average. Additional in the ASes, success rates were calculated assuming that it does not take any time to send data or pass logs however, this was not the case. So the average values were not accurate. This could have been prevented if a timer was used and every AS computed averages based on the same time frames. It could make the things much clearer if baseline traffic measurements were taken in a much earlier stage of the exercise.

Although our network topology was relatively simple to other networks, we were able to handle larger traffic volume and our attack did not disrupt the services. It would have been nice to see how the IPTables handled traffic if the router was not the bottleneck.

H. CONCLUSION

We built a simple administrative domain with some of the common servers with security in mind. Then layered security measures were put in place and we also attacked another AS. Through this exercise we realized that even protecting a simple network is not easy, even if you have all the defenses in place. Theory is different from the reality (as shown by our router) there is always a possibility of an unexpected vulnerability.

References

[1] [1] A Kuzmanovic and E W Knightly. Low-Rate TCP-targeted Denial of Service attacks (The Shew vs. the Mice and Elephants), SIGCOMM 2003, Germany, Aug. 2003.

[2] ntop – Network top, http://www.ntop.org

[3] The Multi Router Traffic Grapher (MRTG) - http://oss.oetiker.ch/mrtg/

[4] Snort - http://www.snort.org/

[5] Packet toolkit (Packit) - http://www.intrusense.com/software/packit/
[6] Nessus vulnerabilit y scanner - http://www.nessus.org/
[7] Wireshark - http://www.wireshark.org/
[8] TCPDump - http://www.tcpdump.org/
[9] IPTables - http://www.netfilter.org/
[10] Unicast Reverse Path Forwarding – http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/uni_rpf.htm