Global Information Assurance Certification Paper

Subject: **SANS – GIAC Certified Intrusion Analyst (GCIA)**

Practical Assignment V3.3

Final - V1.5

date: **June 3rd, 2003**

from: **Loic Juillard**
AT&T Labs
858-795-5554
loic@att.com

INDEX

REVISIONS

Authors	*Release date*	*Comments*
Loic Juillard	**06/03/03**	**Initial document**

AUTHOR(S)

Author	Phone	Email	Role	Sections
Loic Juillard	**858-795-5554**	**juillal@attens.com**	**Student**	

PART 1: STATE OF INTRUSION DETECTION

Virtualized IDS technology

1 Method and apparatus presentation

1.1 What is a virtualized architecture?

"In all large corporations, there is a pervasive fear that someone, somewhere is having fun with a computer on company time. Networks help alleviate that fear." This famous quote from John C. Dvorak illustrate today's telecommunication industry dilemma. As the scale of our IP business increase, the complexity to manage its resources increases exponentially. At this early stage of the networking technology: scaling an IP local or wide area network remains a challenge.

As of today, most of the networking equipment remains dedicated to canonical functions; distinct equipment are specialized into routing, switching and analyzing the traffic into a single local area network. Each one of those devices has to operate within their own, isolated context in a single deployment instance. This architecture translates into overhead costs for purchasing and operating the overall system and high troubleshooting complexity offset by more elaborated support platform to give a uniform front end view to heterogeneous equipment. As the scale of our offers increase automation becomes critical to sustain a quality of services and preserve the foundations of our business.

1.2 What is the use for a virtual architecture?

Inkra claims to be able to virtualizes and integrate multiple IP services including firewall, load balancers, SSL accelerators, VPN, and web accelerators in a single system with the integrity of dedicated appliances and economic and operational benefits of a single system. The aim of this document is to assess Inkra's IDP solution in term of features and performance. Our test will focus on test Inkra's ability to analyse traffic at high speed and determine the limiting factor of this type of architecture. We will also perform a basic alarm detection and stream reassembly testing to prove Inkra's IDP detection capabilities.

1.3 The Inkra IDS module: IDP (Intrusion Detection and Protection)

The Inkra IDP is a based on the implementation of the famous network based IDS "snort". The IDP supports all the snort functionalities as a virtual service module:

- Ships with 1600 predefined attack signatures
- New signatures provided by CERT in Snort format for low maintenance cost
- Includes port-scan anomaly detection to detect network reconnaissance

• Provides a GUI based management through Inkra's Center Point management suite

The IDP VSM works by collecting information at strategic vantage points, typically behind a firewall, between the front end and the server farm. The IDP analyzes the information against criteria defined in its security policies, and if an attack is detected may initiate a number of responses, including:

• Logging the attack
• Alerting by generating a system alarm
• Preventing intrusion by taking an action such as dropping the session

Security policies are applied to interfaces. The security policy of an interface is made up of a collection of rules. These rules specify the criteria against which incoming traffic is to be matched, as well as the kind of logging and alerting that is to take place, and the kind of action that is to be taken. Rules specify the source and destination IP addresses that are to be matched in packets. In addition, rules are based on *attack signatures*, which specify in detail the attributes of the traffic that is to be watched for. Attack signature information includes service (protocol and port) information, the direction of traffic flow, packet options (such as TCP flags, payload size, or protocol anomalies) to watch for. The IDP VSM can also search within the contents of packet payloads, examining them for well-known patterns of attack. This allows you to spot dangerous payloads before they reach their destination. For most of the known attacks, the signatures are provided by the CERT in the snort format and can be imported directly in the IDP VSM. The user can also customize the signatures for customized signature monitoring.

As a virtual module, Inkra's IDS implementation cannot be setup in "tapped" mode, the IDP module is in-line with the rests of the modules which can be a major issue should the IDS have a vulnerability.

2 Hardware overview

2.1 The Inkra 4000

The Inkra 4000 chassis contains 14 front-access vertical slots. The center two slots are reserved for management cards (Switch Management Modules). The remaining twelve slots are universal slots, and can be used for any combination of I/O and processing cards, provided that at least one I/O module and one processing module are installed. Switch Management Module slots are numbered X1 and X2. The universal slots are numbered 1 through 6, and 9 through 14.

Cards are installed from the front of the chassis. Cable management for the cards is accomplished by means of cable management loops located above the hardware module slots. A standard 19" rack can hold up to two chassis and power supply units.

Inkra provides three types of blades:

- **IOM**: network interfaces

- **SMM**: Switch management module, managing the IP traffic switching
- **SPM**: service processing module: providing processing power for the virtual service modules

All the blade are interconnected by a linear bus capable of 80Gbs of traffic

2.1.1 The service processing module (SPM)

The service processing module provides processing power to the VSM. Each service processing module contains several service processing elements. In the current version each SPM contains 2 SPEs. Each SPE contains a VRP, memory, 1 CPU and cryptographic acceleration. Each SPE are involved in any of the virtual service module provisioned on the chassis. The processing power capacity of the SPM is allegedly equally distributed on all the SPEs available in the chassis and is therefore proportional to the number of SPEs.

The virtual rack processor is the central element in the SPM. The VPP coordinates the different elements on the SPM and decides on which path the data will follow to perform the requested operations. Currently, data can follow two distinct paths on the SPE:
- Slow path: the packet are processed by the CPU for further analysis
- Fast path: the packet is directly sent on the next hop interface and does not need any advanced analysis.

The choice of path is directly conditioned by the type of VSM used within the VR.

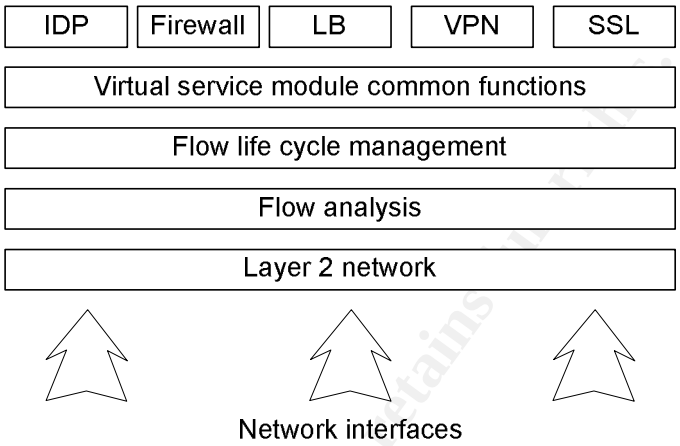
As we mentioned earlier, the data can be processed at the SPE level following two different paths. Once the packet reach the virtual rack processor, if the source IP, destination IP, source port and destination port are already known and if the VR provisioned does require any extensive analysis the data will be able to follow the fast path to lower the latency. On the other hand, if the packet does not meet one of the previous requirements, the packet will be sent in the slow path and be handled by the CPU. You will find bellow a diagram with the different steps on each path.

2.1.2 Switch management module (SMM)

The main function of the switch management module is to coordinate all the chassis function. The SMM also provides a 100BASE-TX management interface for in-band connection and data exchange with the central point server as well as two Card bus readers for flashcard removable storage. The SMM are the only modules to have a dedicated port labeled X1 and X2 on the chassis. The chassis can handle up to two redundant SMM in a hot-standby mode. A single SMM is enough for the chassis to function at maximum capacity.

2.2 Embedded software architecture

Inkra is using a layered architecture divided between the hardware and software. Most of the basic network functionalities are being handled at the ASIC level. The advanced processing of the data are being done on the SPE with a 2 tier software architecture with all the common function implemented in a library and specific functionalities programmed at the upper level.



Inkra embedded software architecture

Unlike dedicated equipment, the Inkra 4000 OS does not consider a packet as a canonical entity. Instead, the OS uses the notion of flows to manage the traffic. A flow of data is characterized by the source IP address, destination IP address, source port and destination port and is assigned based on a static hashing of the 4 parameters to an existing SPE.

2.3 Center point management suite

2.3.1 CLI

The Inkra CLI is organized into a number of contexts, so that you can work with individual devices, organizations, templates, virtual racks, and Ops Link Adaptors. Inkra CLI contexts provide a structure within which families of commands can be grouped, and where you can work in the context of certain specified information in isolation from other system information.

Each Virtual Service Module has its own context, which supports its own command set. Inkra CLI is comparable to CISCO CLI in its functionalities and structure: each context has two modes of operation: exec and configuration. The user can also recall past commands and complete partial commands automatically. The CLI is accessible via SSH or Telnet on the CPS server or directly to the VSS. All the command performed are logged and reported in the audit file.

2.3.2 Web administration

The web administration portal is a central administration platform and behaves as a jump server for all the virtual rack installs within a domain. Each user is created with access privileges centrally on the system and authenticated via password or remote authentication system (radius or LDAP). Each action such as configuration change or passive operations are nominative recorded in a central database and can be accessed at any time to audit past or ongoing operations.

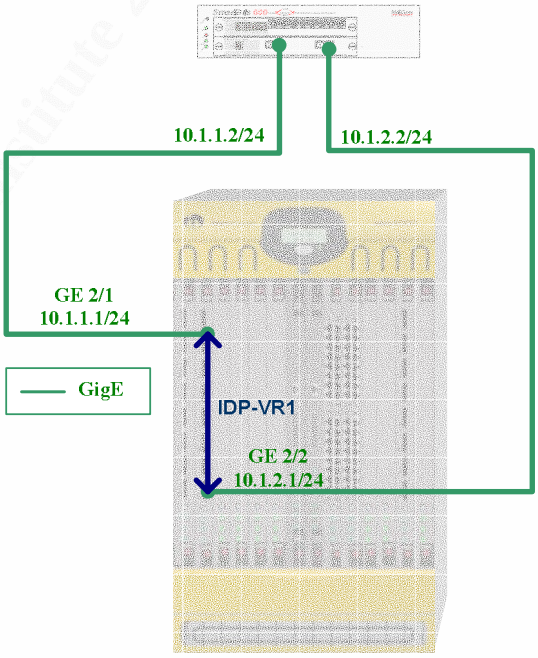
The Web interface gives a graphical access to each VSS parameters, context of the CLI and status as well the provisioned, pending provisioning and deactivated virtual racks. Like the CLI the access is authorized on a command basis.

It is possible to restrict the access to each component on a user or organization basis down to the virtual service module
level.

3 Platform testing

Inkra's IDP version 1.1.1 is the very first public version of their IDS. Our test will be lead in two different parts. We will first assess the equipment intrinsic performances. We will next test the system implemented features and in particular the frame reassembly process.

3.1 Troughput



IDP testing physical architecture

Inkra's VR will be setup as followed:

VR architecture

Note that the VR will be setup with a burst rate of 4 which means that the VR will use all the resources available on the chassis to process the data. The resource management system described above will therefore disregard any SLA setup in the Hardwall parameters.

3.2 Test results and analysis

Inkra claims being able to provide IDS services at "multi-gigabit" speed with close to real time logs. Snort has been known to not be an efficient at high speed mostly because of the numerous false positive the system alarmed on and the resources (processing power and memory) required to reassemble and process ISO/OSI layer 7 data at gigabit speed.

We will start with the signature number testing to assess the performance and how resourceful is the IDS. We will next perform a transactional throughput testing with the full rule base to determine the behavior on normal and high throughput conditions.

For this test we used Whitehats signature file available here:
http://www.whitehats.com/ids/vision.conf.gz
(Tagged with the following export date: "Export date: 20010821.1453")

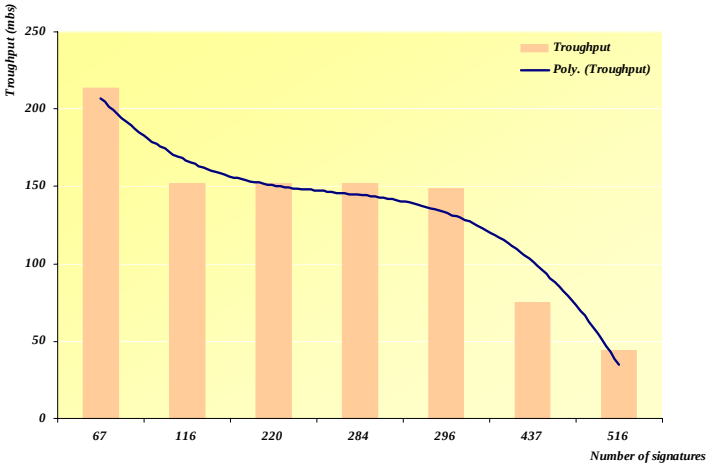
Our choice of signature file was motivated by the signature file structure itself. The whitehats signature file is composed of 544 signatures structured as followed:

- Signatures 1 -> 175 specific exploits with content check
- Signatures 176 -> 254 specific exploits without content check
- Signatures 255 -> 485 vulnerabilities with content check
- Signatures 486 -> 544 vulnerabilities without content check

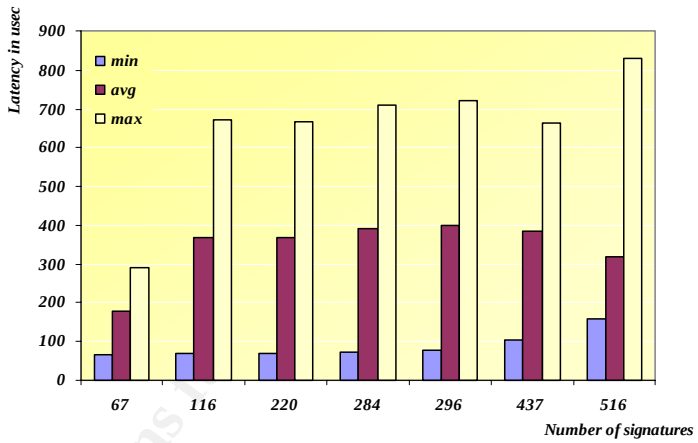
($INTERNAL and $EXTERNAL will be setup as any to force each rule to be executed)

By stripping the signature file by the bottom we will be able to observe how impacting content checking is on the maximum throughput.

For the maximum throughput test we will setup a 3 way handshake TCP connection on port 80 on the Smartbits, transfer 1460bytes of data and close the TCP connection with a standard FIN/ACK. The throughput is the amount of data that have been transferred trough the Inkra IDP. We will assume that we reached the limit when the data loss is higher than 5% of the total data sent.



IDS - Max throughput vs. number of signatures



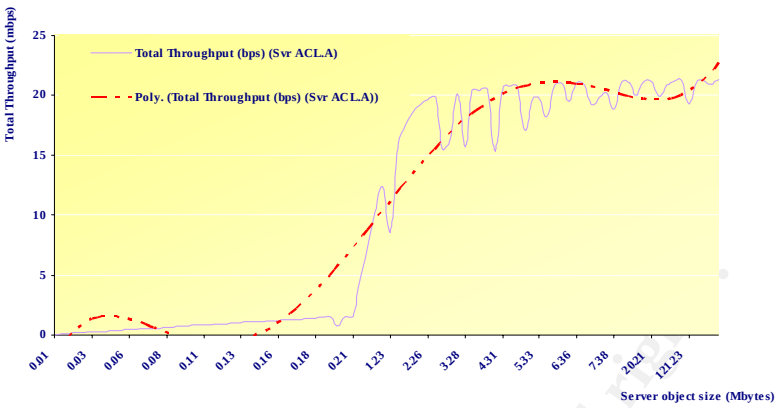
IDS - added latency per number of signature

As we anticipated, the IDP is very resources intensive. On four SPEs and with a burst rate set to 4 (maximum capacity available) the Inkra chassis was not able to push more than 45mbs of traffic with 526 signatures. Most of the processing power is used to perform packet content checking and snort pre-processor reassembly. Surprisingly, the content checking does not seem to have much effect on the IDP, we were expecting to see a tremendous throughput improvement when we removed the content based rules. Content matching and processing power does not seem to be the limiting factor here.

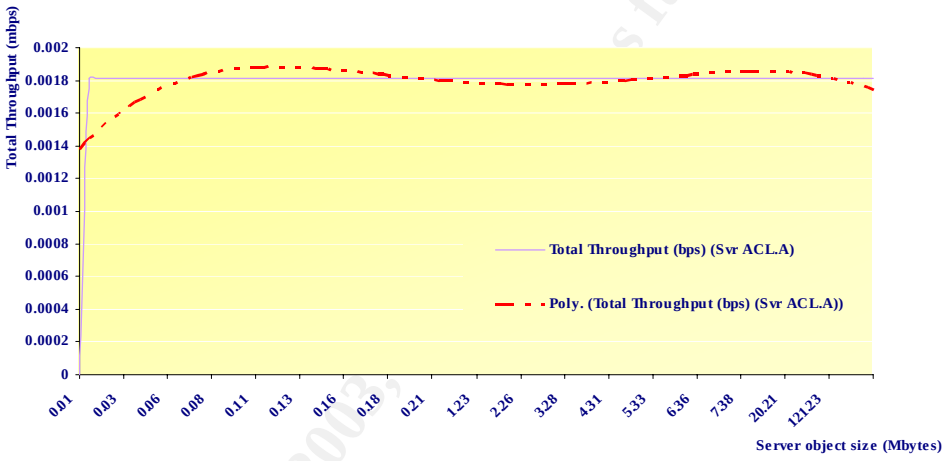
With a burst set to 0, the IDS with 516 rules would not have been able to use more than 10mbs of traffic. Most of the performance issues we are seeing are due to the fact that Inkra has not implemented the snort analysis signature ordering optimization. Instead, each packet is analyzed and compared against each entry in the signature file. We will now determine what resource is maxed out first. We will determine bellow the limiting factor.

For this test we will analyze the behavior of the IDP when the max throughput is reached for an HTTP connection with 516 signatures. A client located on the external side of the IDP will attempt to generate as many connections as it can to an HTTP server in the internal segment in 1 minute. The HTTP server will reply with an incremental amount of data. The bandwidth shown on the vertical axis represents the server side bandwidth only. The client request is not included here.

Note that the Smartbits has been tested for more than 500mbs of bandwidth and is therefore not the limiting factor here.



IDS – full HTTP data transfer server throughput with 526 signatures



IDS – full HTTP data request client throughput with 526 signatures

As we see in the HTTP server graph, the maximum bandwidth for this test is around 20mbs, way lower than what we found in the first test. As we mentioned in the hardware description, Inkra distributes the traffic based on the hash on the source IP address, destination IP, source port and destination port. In our test, since we used a single connection, only 1 SPE at the time was processing traffic, decreasing the overall performance.

The HTTP server side graph can be separated in three parts note that the data in the horizontal axis are not linear):

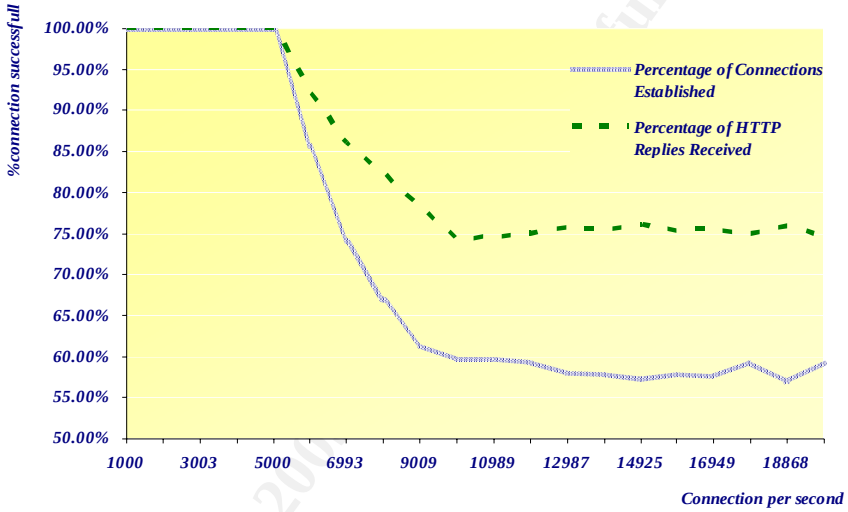
- 0 to 2 Mbytes of data where the bandwidth used is linear
- 2Mbytes and above: the IDP reached the maximum bandwidth, the CPU (gathered via the CLI: show cpu) shows a 100% utilization.

The processing power is the limiting factor for this test. The CPUs were maxed out at 20 Mbs. Since the Inkra 4000 chassis is blade based and that the flow are equally distributed across each SPE, we can assume that adding more SPM to the chassis would

improve the performances. With 12 SPM the chassis would therefore be able to handle 240Mbs of traffic with 516 signatures.

Unfortunately, our equipment limited the test to a single IP. A real life test would involved more source and destination IP addresses. The memory could be the limiting factor then.

We will now examine the behavior of the IDP when the limit is reached. We will setup the Smartbits to generate TCP connection (normal 2 way handshake) on port 80 to the internal side of our architecture and attempt an HTTP request the server will then send 40bytes of data back to the client to signify as a reply. We will measure the amount of HTTP reply received vs. the number of HTTP reply sent.



IDS – full HTTP data request client throughput with 526 signatures

As we can see in this graph, when the IDS resources are maxed out, the established connections (green on the graph) are affected by the lack of processing resources. We noticed that at no point the VR did not generate an alarm when the resources of the SPEs were maxed out. Instead the ISP keeps trying to analyze new flows even if no resources are available.

3.3 Feature testing: detection and alarming

3.3.1 Detection

Next we will test the IDS detection capabilities. We will generate various types of attacks from our Linux RedHat 7.0 client running fragroute v1.2 to a linux RedHat 7.0 server running apachev1.3.38. For each attack we will first record if the attack was detected properly by checking for false positive, false negative and misleading alarms. Next we will analyze the logs to see if it contains enough data for further analysis. Since Inkra's IDP is based on snort we will not perform an extensive IDS detection accuracy. We will perform

several basic test: Syn" scan, "Syn Fin" scan, X-mass tree scan to verify the IP/TCP header and content analysis and an advanced check issuing a common IDS evasion technique using traffic fragmentation.

We will be using the same signature file used in the performance analysis.

- Syn scan:

We used for this test *synscan*, a popular tool to scan ip addresses with in its early version a static IP_ID of 19104 making it easily detectable on an IDS.

Warning	2003/03/24 16:42:56	SPE 11/1	Attack detected. [**] [258][Inkra ID:: 0]IDS521/scan_probe-Synscan-Portscan-ID-19104 [**] [Classification:][Priority: 1] 03/24-16:42:56.000000 10.1.1.2:51128 -> 10.1.2.2:80 TCP TTL:9 TOS:0x0 ID:19104 IpLen:20 DgmLen:44 ******S* Seq: 0xD00A04E5 Ack:0x0 Win: 0x2710 TcpLen:24 2F 20 00 / .

The IDS detected the attacks properly. Note that the content of the packet is displayed in the log. This feature is very interesting for further forensic analysis and accurate correlation. Based on these logs it is possible to perform additional off line analysis on the packet (typically by a correlation tool like Intellitactics) to filter any potential false positive, passive OS fingerprinting, alarm reduction…

- Syn-fin scan:

We scan our server using hping2 2.0.0 RC2 with following command line:

```
Hping2 –S –F 10.1.2.2 –p 80 –d 90
```

Warning	2003/03/25 8:49:17	SPE 4/1	Attack detected. [**] [499][Inkra ID:: 0]IDS198/scan_SYN FIN Scan [**] [Classification:][Priority: 1] 03/25-08:49:17.000000 10.1.1.2:1025 -> 10.1.2.2:80 TCP TTL:63 TOS:0x0 ID:46566 IpLen:20 DgmLen:110 ******SF Seq: 0x1E240 Ack:0x0 Win: 0x2710 TcpLen:20 00 00 00 00 00 00 FF FF FF FF FF FF 00 00 00 00 ..... 00 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... 00 00 00 00 00 03 20 01 00 02 00 00 00 17 DE 41 ....A 44 CA 21 BE BB 35 20 D.!..5

The IDS detected and logged the attack properly.

3.3.2 Packet reconstruction

Attackers usually use a combination of techniques to get around the IDS detection engine. For this test we choose a basic signature involving the content of a packet (snort style signatures):

alert ICMP $EXTERNAL any -> $INTERNAL any (msg: "IDS152/icmp_Ping BSDtype"; itype: 8; content: "|08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17|"; depth: 32;)

We then used a packet crafter (hping2) on the client side and generated 8bytes segmented ICMP traffic towards our internal server with fragrouter. The IDS detected successfully the *"08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17"* pattern in the ICMP payload event though the payload was fragmented into 2 separate packets. The IDS is able to reassemble up to 100 IP packets within a single connection.

.

Warning	2003-03-26 11:35:24.614	SPE 4/2	Attack detected. [**] [84][Inkra ID:: 0]IDS152/icmp_Ping BSDtype [**] [Classification:][Priority: 1] 03/26-11:35:24.000000 10.1.1.2 -> 10.1.2.2 ICMP TTL:63 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF Type:8 Code:0 ID:42357 Seq:512 ECHO 08 00 34 20 A5 75 02 00 64 A1 80 3E 4C 87 00 00 ..4 .u..d..>L... 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 ..... 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 ..... !"#$%&' 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 ()*+,-./01234567 00

We noticed that the packet did not reach the server. The VR dropped the packet because the fragment was too small:

Warning	2003/03/26 12:04:24	SPE 4/2	IP fragment attack: Interrior Fragment Too Small. Src 10.1.1.2; Dst 10.1.2.2; Protocol 6

The Inkra SPE rejected the packet because the fragment was too small. The vendor indicated later on that fragments smaller than 96bytes would be rejected internally and that the user could not change this behavior.

3.4 Reports

In addition to the alarms listed above, the IDP via the central point server provides a GUI access to the IDP configuration at the VSM level. The interface allows the user to manage:

- Signatures (addition, removal, enable, etc…)
- attack alert level
- Log configuration
- scanning detection parameters
- monitoring addresses (internal segments, external segments, etc …)

The central point server access also provides graphical access to monitoring data:

- Number of attacks per category
- Number of attacks per interfaces
- Number of attacks per policy
- Amount of traffic dropped
- Amount of traffic analyzed

4 Conclusion

Due to the performance issues we have seen, the IDS would not be able to operate as a bastion IDS (IDS installed in front of a firewall on the WAN interface). The Inkra IDP VSM would perform optimally in conjunction with a firewall and with a signature list restricted to the traffic allowed through the firewall. The IDS will then perform the signature matching only on allowed inbound traffic rather than checking all the incoming traffic. This architecture is used today by our customers and is part of the standard offer.

The IDS needs to be configured in-line with the rest of the services, no tapping is allowed here. The IDS impact the traffic negatively by adding a significant amount of latency and jitter on the monitored traffic.

On the other hand the IDS provided an efficient GUI and graphical reports on the alarms generated.

IDS	
Rules in SNORT format for low cost rules addition Good logging (snort) Good alarming (snort) Good administration interface	Performance Poor performance as a front end IDS: need to be behind a firewall Traffic latency is being affected by the IDS (latency multiplied by 5 compared to layer 3 switching)

The following comparative analysis displays the throughput results for each platform with approximately same testing parameters with 250 rules (see references for data source):

Inkra IDP with 12 SPM	Ipenforcer 5000
1500Gbs*	2Gbs

* our test showed 250mbs with 230 rules and 2SPMs. Interpolated for 12SPM assuming a linear growth.

5 Resources and references

[1] Network World Fusion, 05/12/03, "*Why IPS products haven't taken off*" - Ellen Messmer
http://napps.nwfusion.com/weblogs/security/002755.html

[2] The Tolly group, January 2002, "iPolicy Networks ipEnforcer 5000"
http://www.ipolicynetworks.com/pdf/tolly_report.pdf

[3] The Tolly group, 8/2/2002, "*Inkra Networks Corp. Inkra 4000 Virtual Service Switch HardWall Technology Evaluation*"
http://www.tolly.com/DocDetail.aspx?DocNumber=202145

[4] Miercom, December 2002, "*SecureNet 7145C benchmark*"
http://www.miercom.com/dl.html?fid=20021206&type=report

[5] computer.org, 1996, Nicholas J. Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, Ronald A. Olsson, "*A Methodology for Testing Intrusion Detection Systems*"
http://www.computer.org/tse/ts1996/e0719abs.htm

[6] Neohapsis, July 2002, *ISS RealSecure 7.0 IDS testing*
http://osec.neohapsis.com/results/nids/realsecure-7.0-08.15.2002/index.html

[7] Neohapsis, July 2002, *IntruVert Intrushield 0.97.12*
http://osec.neohapsis.com/results/nids/intrushield-0.97.12-08.17.2002/index.html

[8] Neohapsis, February 2003, *Netscreen IDP-100 2.0p2-2*
http://osec.neohapsis.com/results/nids/intrushield-0.97.12-08.17.2002/index.html

[9] Neohapsis, May 31 2003, *NFR NID-310 3.2*
http://osec.neohapsis.com/results/nids/intrushield-0.97.12-08.17.2002/index.html

[10] Neohapsis, Nov 11 2002, *Sourcefire NS3020F 2.6.0*
http://osec.neohapsis.com/results/nids/sourcefire-ns3020f-2.6-06.25.2003/index.html

[11] ISECOM, *OSSTMM - Open Source Security Testing Methodology Manual*
http://www.isecom.org/projects/osstmm.htm

PART 2: NETWORK DETECTS

Virtualized IDS technology

Attack number 1:2002.5.29 – A TCP de-synchronization attack

1. Source of trace: incidents.org URL: www.incidents.org/logs/raw/2002.5.29

We will focus on the traffic to and from 46.5.180.250. The rests of the logs will be used to determine the network topology and equipment roles. First we will try to determine where the IDS is located on the network. The following command will help us determine the MAC address on each side of the IDS:

```
> Tcpdump -neqr 2002.5.29 | cut -d ' ' -f2,3 | sort –n | uniq
    00:00:0c:04:b2:33 00:03:e3:d9:26:c0
    00:03:e3:d9:26:c0 00:00:0c:04:b2:33
```

Both OUI (Organizationally Unique Identifier: first three-octets of the MAC address) are assigned to CISCO. However, we have not been able to determine the exact interface model associated with the OUI. Based on those MAC we determined that the IDS was probably tapped between two CISCO routers 00:03:e3 being the uplink. 00:00:0c would then be the next hop to reach 46.5.180.250. (see diagram bellow).

Let's now take a look at 46.5.180.250. To make things easy we will name this equipment '*C'*. In all the traces, P seems to be a client trying to download information from various servers on the port (destination port) 80/TCP (port commonly used for HTTP). As we examined the content of the packets we noted that the HTTP "User-agent" tag varied between the sessions:

- in the frame 220 the client was listed as MSIE5.5; Windows 95
- in the frame 300 the client was listed as MSIE 5.5; windows NT 5.0

```
Frame 220 (629 bytes on wire, 629 bytes captured)
Ethernet II, Src: 00:00:0c:04:b2:33, Dst: 00:03:e3:d9:26:c0
Internet Protocol, Src Addr: 46.5.180.250 (46.5.180.250), Dst Addr: 209.225.0.6 (209.225.0.6)
Transmission Control Protocol, Src Port: 64132 (64132), Dst Port: http (80), Seq: 494238966, Ack:
1465705210, Len: 575
 Hypertext Transfer Protocol
    GET /site=72385/size=468060/bnum=%n/bins=1/rich=0 HTTP/1.1\r\n
        Request Method: GET
    Accept: */*\r\n
    Referer: http://www.accuweather.com/adcbin/isight_video?nav=video&partner=yahoo&city=nyc\r\n
    Accept-Language: en-us\r\n
    Accept-Encoding: gzip, deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 95)\r\n
    Host: servedby.advertising.com\r\n
    Connection: Keep-Alive\r\n
    Cookie:  170.129.50.120=-e11e,3d128fa4-;  28709328=-d417,3d128f89-;  84218867=-e0ca,3d128f9c-;
51412440=-e11e,3d128fa1-;            %n=-c8ae,3d1d1f08,,-;            ACID=ee820010159030120002!;
BASE=kPL/qgbnv+rs+MkUrvcWaVDbZMEnOWlavsB+kwL1h+mPbN2q4WC2nTHBOAv4JIF!\r
    \r


Frame 300 (524 bytes on wire, 524 bytes captured)
Ethernet II, Src: 00:00:0c:04:b2:33, Dst: 00:03:e3:d9:26:c0
Internet Protocol, Src Addr: 46.5.180.250 (46.5.180.250), Dst Addr: 208.184.29.90 (208.184.29.90)
Transmission Control Protocol, Src Port: 61729 (61729), Dst Port: http (80), Seq: 2472303082, Ack:
203814497, Len: 470
Hypertext Transfer Protocol
    GET /adi/N3139.NY_Times/B987199;sz=550x550;ord=%%GMTTIME%%? HTTP/1.1\r\n
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword, */*\r\n
    Referer: http://www.nytimes.com/ads/beth/CarrotInk.html\r\n
    Accept-Language: en-us\r\n
    Accept-Encoding: gzip, deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)\r\n
    Host: ad.doubleclick.net\r\n
    Connection: Keep-Alive\r\n
    Cookie: id=800000152321a27\r\n
    \r\n
```

We also noted that most of the packet logs were targeting 64.154.50.51 with a different hitbox.com cookie number (CTG=XXXXXX) in a short period of time (less than 5min.). Based on those observations, we guessed that P is actually an HTTP proxy (or SOCK server) used by an organization to get access to the internet. Note that since the user-agent tag is easily “spoofable” this last information might not be true. We will call *‘S’* the remote server.

As we tried to determine the type of proxy server using passive fingerprinting technique, we saw that within a same TCP session packets had very distinct parameters. The first category of packets (category 1):

- had the identification number set
- Had the DF bit set
- A TTL of 124
- A window size of 17520

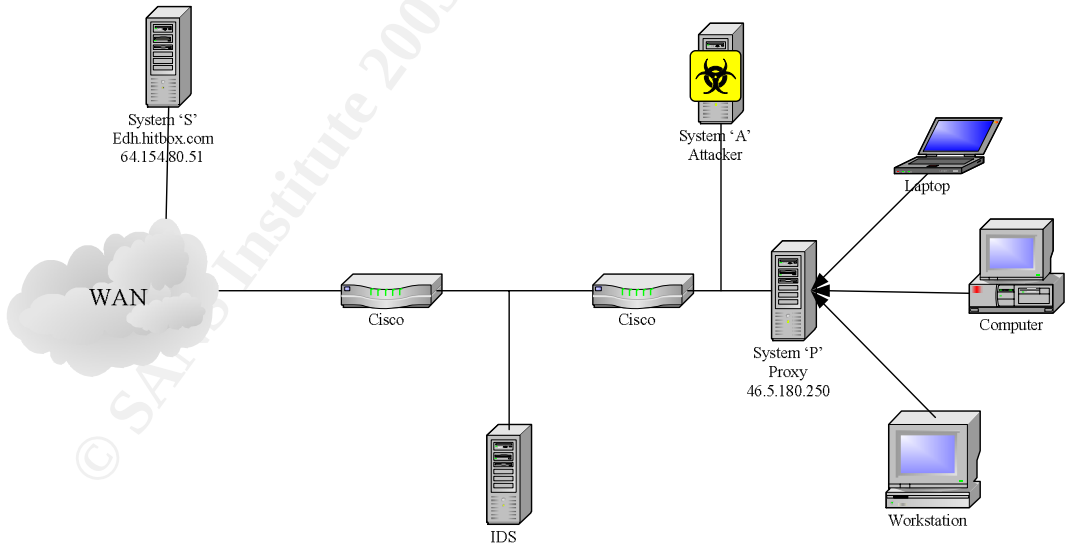
```
Frame 248 (278 bytes on wire, 278 bytes captured)
Ethernet II, Src: 00:00:0c:04:b2:33, Dst: 00:03:e3:d9:26:c0
Internet Protocol, Src Addr: 46.5.180.250 (46.5.180.250), Dst Addr: 64.154.80.51 (64.154.80.51)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 264
    Identification: 0x3195 (12693)
    Flags: 0x04
    Fragment offset: 0
    Time to live: 124
    Protocol: TCP (0x06)
    Header checksum: 0x5e94 (incorrect, should be 0x588e)
    Source: 46.5.180.250 (46.5.180.250)
    Destination: 64.154.80.51 (64.154.80.51)
Transmission Control Protocol, Src Port: 61524 (61524), Dst Port: http (80), Seq: 2047509814, Ack:
4230865524, Len: 224
    Source port: 61524 (61524)
    Destination port: http (80)
    Sequence number: 2047509814
    Next sequence number: 2047510038
    Acknowledgement number: 4230865524
    Header length: 20 bytes
    Flags: 0x0018 (PSH, ACK)
    Window size: 17520
    Checksum: 0x5ff6 (incorrect, should be 0x59f0)
Hypertext Transfer Protocol
    Data (224 bytes)
```

According to passive fingerprinting table, this system would be identified as a Windows 2000 server. However, the second category of packet has (category 2):

- No identification number set
- did not have a DF bit set
- A TTL of 240
- A window size of 32120 or 33580

```
Frame 247 (349 bytes on wire, 349 bytes captured)
Ethernet II, Src: 00:00:0c:04:b2:33, Dst: 00:03:e3:d9:26:c0
Internet Protocol, Src Addr: 46.5.180.250 (46.5.180.250), Dst Addr: 64.154.80.51 (64.154.80.51)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x10 (DSCP 0x04: Unknown DSCP; ECN: 0x00)
    Total Length: 335
    Identification: 0x0000 (0)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 240
    Protocol: TCP (0x06)
    Header checksum: 0x0000 (incorrect, should be 0x55cc)
    Source: 46.5.180.250 (46.5.180.250)
    Destination: 64.154.80.51 (64.154.80.51)
Transmission Control Protocol, Src Port: 61474 (61474), Dst Port: http (80), Seq: 2787956848, Ack:
2013235585, Len: 295
    Source port: 61474 (61474)
    Destination port: http (80)
    Sequence number: 2787956848
    Next sequence number: 2787957143
    Acknowledgement number: 2013235585
    Header length: 20 bytes
    Flags: 0x0018 (PSH, ACK)
    Window size: 33580
    Checksum: 0x0000 (incorrect, should be 0x07b6)
Hypertext Transfer Protocol
    Data (295 bytes)
```

Some systems (like Linux) use an IP_ID of 0 in many cases where the "Don't Fragment" bit is not set. So this system might also very well be a Linux system.



2. Detect was generated by:

Detect was generated by snort stream4 pre-processor. As we examined Stream4 alert message, we determined that the alert was likely:

"(spp_stream4) Possible RETRANSMISSION detection"

It might also be:

```
"(spp_stream4) WINDOW VIOLATION detection"
```

Those alarms are turned off by default on the preprocessor.

We used *tcpdump* and ethereal to perform the analysis.

3. Probability the source address was spoofed:

Considering the first observation we did in the first section, we can assume that P address has been spoofed. As we mentioned earlier we have within each session two types of packets, the question is: which ones (if any) are really coming from C? To determine this we will need to find packets with erroneous parameters or anomalies. Let's assume that we are using Ethernet (very common) as a data link mechanism. The frame #3 (a category 2 packet) is telling us that the packet had 4434 bytes on the wire, virtually impossible considering that the maximum size for an Ethernet Frame is 1500bytes (unless the network uses Gigabit Ethernet jumbo frames limited to 9000 bytes). We also noticed that we had several occurrences of duplicated data: a category 1 packet flowed immediately by a category 2 packet (215 and 216, 217 and 218, 300 and 301…). Our guess is as this point that the category 2 (Linux server) packets are being sent from a reactive system after a category 1 (windows server). Our hypothesis is that a compromised/evil system is sending spoofed packets (category 2 packets) based on the information/state of the real client C. We will name the system sending the spoofed category 2 packets *'A'*. Let's now try to answer to the Why?

4. Description of the attack:

The attack consists in a set of ACK packet sent by and internal system to remote system with bogus sequence and ACK number.

A analysis of the packet show that within a same session the identification number the DF bit set, the TTL and window parameters belong to what appears to be two distinct systems.

5. Attack mechanism:

The first question we will answer is "Stimulus or response"? According to the pattern we observed above, we are looking at a response from A to a stimuli sent by P.

Bellow is the traces of 4 packets belonging to the same TCP session (same source IP/port same destination IP/port in less than 30s interval):

```
Frame 1 (1514 bytes on wire, 1514 bytes captured)
    Arrival Time: Jun 28, 2002 17:00:55.134488000
    Time delta from previous packet: 0.000000000 seconds
    Time relative to first packet: 0.000000000 seconds
    Frame Number: 1
    Packet Length: 1514 bytes
    Capture Length: 1514 bytes
Ethernet II, Src: 00:00:0c:04:b2:33, Dst: 00:03:e3:d9:26:c0
Internet Protocol, Src Addr: 46.5.180.250 (46.5.180.250), Dst Addr: 64.154.80.51 (64.154.80.51)
Transmission Control Protocol, Src Port: 61013 (61013), Dst Port: http (80), Seq: 555149661, Ack:
1599014187, Len: 1460
Hypertext Transfer Protocol
    Data (1460 bytes)

Frame 2 (971 bytes on wire, 971 bytes captured)
    Arrival Time: Jun 28, 2002 17:00:55.244488000
    Time delta from previous packet: 0.110000000 seconds
    Time relative to first packet: 0.110000000 seconds
    Frame Number: 2
    Packet Length: 971 bytes
    Capture Length: 971 bytes
Ethernet II, Src: 00:00:0c:04:b2:33, Dst: 00:03:e3:d9:26:c0
Internet Protocol, Src Addr: 46.5.180.250 (46.5.180.250), Dst Addr: 64.154.80.51 (64.154.80.51)
Transmission Control Protocol, Src Port: 61013 (61013), Dst Port: http (80), Seq: 1599022947, Ack:
555149661, Len: 917
Hypertext Transfer Protocol
    Data (917 bytes)

Frame 3 (4434 bytes on wire, 1514 bytes captured)
    Arrival Time: Jun 28, 2002 17:00:55.344488000
    Time delta from previous packet: 0.100000000 seconds
    Time relative to first packet: 0.210000000 seconds
    Frame Number: 3
    Packet Length: 4434 bytes
    Capture Length: 1514 bytes
Ethernet II, Src: 00:00:0c:04:b2:33, Dst: 00:03:e3:d9:26:c0
Internet Protocol, Src Addr: 46.5.180.250 (46.5.180.250), Dst Addr: 64.154.80.51 (64.154.80.51)
Transmission Control Protocol, Src Port: 61013 (61013), Dst Port: http (80), Seq: 555149661, Ack:
1599021487, Len: 4380
Hypertext Transfer Protocol
    Data (1460 bytes)

Frame 4 (2431 bytes on wire, 1514 bytes captured)
    Arrival Time: Jun 28, 2002 17:00:55.354488000
    Time delta from previous packet: 0.010000000 seconds
    Time relative to first packet: 0.220000000 seconds
    Frame Number: 4
    Packet Length: 2431 bytes
    Capture Length: 1514 bytes
Ethernet II, Src: 00:00:0c:04:b2:33, Dst: 00:03:e3:d9:26:c0
Internet Protocol, Src Addr: 46.5.180.250 (46.5.180.250), Dst Addr: 64.154.80.51 (64.154.80.51)
Transmission Control Protocol, Src Port: 61013 (61013), Dst Port: http (80), Seq: 555149661, Ack:
1599023864, Len: 2377
Hypertext Transfer Protocol
    Data (1460 bytes)
```

We will now describe in describe the attack as seen in the logged packets using the following layout:

Source -> destination: Seq(X), ACK Seq(Y), bytes of data sent [Flags]

Where:

- Source: the source system
- Destination: the destination system
- Seq(X): the initial sequence number for the source
- ACK Seq(Y): the ACK number

- bytes of data sent: the amount of data sent in bytes
- Flags: the flags used (e.g. [PSH, ACK])

The attack would be represented as followed:
1: A -> S: Seq(S), ACK Seq(C), 1460 bytes of data [PSH, ACK]
2: C -> S: Seq(C)+6*1460, ACK Seq(S), 917 bytes of data [PSH, ACK]
3: A -> S: Seq(S), ACK Seq(C)+6*1460+5*1460, 4380 bytes of data [PSH, ACK]
4: A -> S: Seq(S), ACK Seq(C)+ 6*1460+5*1460+1460+917, 2377 bytes of data [PSH, ACK]

We also noticed that the checksum for packet coming from A is set to 0. The REAME file located in the incident.org/raw/logs directory specified that checksums where altered by the "anonymazer" process. We can't tell for sure if the original packets had a checksum set to 0.

We believe the traffic seen here is only the tip of the iceberg. A is attempting to manipulates the ACK and sequence numbers to affect P's traffic.

Let's now go deeper into the analysis. As we mentioned earlier, the traces show only the packets that violated stream4 reassembly rules. Valid packets have not been logged here. We will now try to reconstitute the missing parts and determine the attack mechanism assuming that the logged packets are actually not being dropped.

Since S is a server (hitbox.com web server) and C is a client, we can assume that S and C can establish a connection without any problem.

After C and S completed a successful 4 way handshake, let's try to reconstitute the complete transaction. We will color in red the entries present in the logs and in blue the entries we reproduced based on TCP/IP data transfer mechanism.

1: S -> C: Seq(S), 1460 bytes of data
2: A -> S: Seq(S), ACK Seq(C), 1460 bytes of data
3: C -> S: Seq(C), ACK Seq(S)+1460, 1460 bytes of data (Corrective ACK not received yet)
4: S -> C: Seq(S)+1460, ACK Seq(C)+1460 (Corrective ACK because wrong ACK in #1: #3 is **discarded**)
5: C -> S: Seq(C)+1460, ACK Seq(S)+1460, 1460 bytes of data (Corrective ACK not received yet)
6: C -> S: Seq(C)+1460, ACK Seq(S)+1460, 1460 bytes of data (retransmit after corrective ACK received)
7: C -> S: Seq(C)+1460, ACK Seq(S)+1460+1460, 1460 bytes of data (delayed ACK, send 1460bytes of data)
8: A -> S: Seq(S), ACK Seq(C)+1460+1460, 1460 bytes of data
9: S -> C: Seq(S)+1460, ACK Seq(C)+1460+1460+1460 (Corrective ACK because wrong ACK received)

'A' tries to slow down connections between C and S by injecting bogus packets forcing the hosts to resync their sequence numbers. The disturbance caused by A in the model above is 1 data packet dropped for 2 data packet sent by the client or about 33% of traffic impacted.

This attack can be implemented using simple UNIX scripting tools with for example SPAK and tcpdump. Since the hacker did not want to generate random sequence number, he basically swapped the sequence and ACK number sent by C. This technique helped us to evaluate the real impact on the traffic in a single transmission with the example listed in section 4:

1: A -> S: Seq(S), ACK Seq(C), 1460 bytes of data [PSH, ACK]
2: C -> S: Seq(C)+6*1460, ACK Seq(S), 917 bytes of data [PSH, ACK]
3: A -> S: Seq(S), ACK Seq(C)+6*1460+5*1460, 4380 bytes of data [PSH, ACK]
4: A -> S: Seq(S), ACK Seq(C)+ 6*1460+5*1460+1460+917, 2377 bytes of data [PSH, ACK]

We can see that the client sequence number that according to our theory can be read from A's ACK number is increasing. Actually in 0.22s, 12597 bytes seems to have been transferred or 447K/s of bandwidth. The attack doesn't seem to be efficient at high speed rate. By targeting the proxy server the attacker is expecting to affect more users than if the users had a direct connection.

Coupled with DoS (SYN flood) on the proxy the attacker could slow down the proxy server preventing it to send back the ACK and eventually time-out the connection. According to the network architecture we explained above, the traffic from A to C is likely to not be on the IDS path and therefore would not be monitored by the IDS.

6.Correlation

We haven't been able to find an exact match for the attack; however we found several similar studies:
http://us1.unix.geek.org.uk/~arny/iphijack.txt

And a study on common TCP attacks by CHRIS CHAMBERS, JUSTIN DOLSKE, and JAYARAMAN IYER
http://www.linuxsecurity.com/resource_files/documentation/tcpip-security.html

Another lead might be actually a false positive. Mads Rasmussen reported having the same type of alarm with a proxy server:
http://www.geocrawler.com/archives/3/4890/2001/8/400/6442171/
The site mentioned in that email (snort.sourcefire.com) giving an explanation from Martin Roesch is unfortunately down.

Martin Roesch mentioned that version prior to snort 1.8.4-beta 1 did not handle stream 4 retransmission properly:
http://archives.neohapsis.com/archives/snort/2002-01/0792.html
snort v1.8.4 final was released in May 2002. The logs are time stamped from June 28th 2002. The administrator might be running an older version of snort.

7. Evidence of active targeting:
There is no evidence of active targeting. The attacker only aim was to attempt to slow down the access to what we thought was a proxy may be to get better access to the internet (congested network?). Note that the target here is not the remote server but actually the clients that try to go through the proxy.

8. Severity
S = (5 + 2) – (1+1) = 5

Critically: the target for this attack is the integrity of the internal network as a whole.

Lethality: If the attack succeeded at 100%, the connection could ultimately slow down tremendously to the point where is would actually break.

System countermeasures: there is nothing that can be done at the system level to prevent this since the bogus packets from a remote system.

Network countermeasures: most of the stateful firewall out there to not perform a TCP stream inspection. Once the 3 way handshake occurs the firewall passes the traffic without any distinction until the connection is reset or terminated gracefully.

9. Defensive recommendation

The proxy could be moved to a different network segment isolated from the users and filter spoofed traffic with a proper set of ACL on the users gateway, the attacker would not be able to spoof the proxy address. The network administrator could also sniff the internal network to try to gather the system MAC address and locate physically the server.

10. Multiple choice test question
In an established TCP session, what would a system do if it receives a TCP packet with a wrong ACK numbers?

a. reset the connection
b. discard the packet
c. emit a packet with its current sequence number and ACK number to re-synchronize the connection
d. compare the received IP_ID number with its internal IP_ID number and drop the packet if they are not equal

Answer: c

Explanation: if a host receives a packet with a bad ACK number, the host sends a corrective 'ACK' packet back in order to resynchronize each host and retransmit missing packets.

Questions/Comments:
We received only 2 questions from Don Murdoch regarding our analysis:

Q1: *Hmmm... I think you are in danger again. If this is internet traffic, then you need to explain why from a non-MAC point of view the source is real / spoofed.*
A discussion around MAC addressees would only be applicable on a specific network segment, as a router clouds the issue.

Our analysis specified that what we believe being the "Real" host: P the DF bit is set. There is no evidence that the packets have been fragmented at the source. On the contrary we can say considering the consistent increment of the ACK counter by 1460 that the maximum packet size for the host is actually the MTU of a standard Ethernet frame. Furthermore our analysis is not based solely on packet size: passive fingerprinting and protocol consideration motivated our conclusions.

Q2:

(talking about the C, S and P terms)
I think that if you are going to go off into computer science land than you need to drag out visio or draw pretty boxes in word to actually show who's the C, S, P, and ... Are.

Security and analysis of network activity is a computer science. We used letter to represent system to abstract the problem from any unreliable values such as source IP addresses and destinations IP to attempt to give an explanation to the activity explained here.

Attack #2: 2002.10.17 – ACK Scan

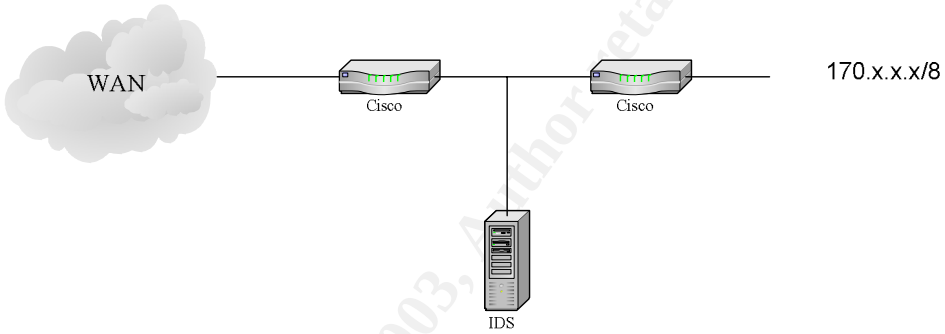
1. Source of trace: incidents.org URL: www.incidents.org/logs/Raw/2002.10.17

I will use a small subset of data in the pcap file with reflective ports ACK TCP packets (destination port = source port) as a starting point for this analysis. Typical TCP connections are coming from an ephemeral port (>1024) to an assigned port (<1024). Therefore any connection coming from a port lower than 1024 is suspicious, even more suspicious if the source port is the same than the destination port (with some exceptions: IKE, DNS, etc..)

We use the rest of the logs to determine if the attack was coordinated, distributed or localized to a single device. First we will try to determine where the IDS is located on the network. The following command will help us determine the MAC address on each side of the IDS:

```
> Tcpdump -neqr 2002.10.17 | cut -d ' ' -f2,3 | sort –n | uniq
   00:00:0c:04:b2:33 00:03:e3:d9:26:c0
   00:03:e3:d9:26:c0 00:00:0c:04:b2:33
```

The MAC addresses are the same than for the detect #1. Both OUI (Organizationally Unique Identifier: first three-octets of the MAC address) are assigned to CISCO. Based on those MAC we determined that the IDS is tapped between two CISCO routers 00:03:e3 being the uplink. 00:00:0c would then be the next hop to the internal segment 170.x.x.x/8. (see diagram bellow).



Let's now focus on the attack itself. Let's try to determine the number of packets with the reflective port pattern and the list of internal and external hosts involved in the reflective port pattern. The following command will give us the number of packets with reflective ports:

```
$ tcpdump –nr 2002.10.17 "tcp[0:2] == tcp[2:2]" | wc -l
36
```

We have 36 reflective port packets logged in this file. The following command will help us to determine quickly the combination of IP addresses used:

```
$ tcpdump -nqr 2002.10.17 "tcp[0:2] == tcp[2:2]" | cut -d ' ' –f 3,5 | sort -n | uniq
61.218.15.118.80 170.129.192.213.80:
61.218.15.126.80 170.129.192.213.80:
61.218.161.202.80 170.129.44.252.80:
61.218.161.210.80 170.129.44.252.80:
61.221.88.198.80 170.129.192.213.80:
61.221.99.242.80 170.129.31.29.80:
61.222.154.109.80 170.129.130.226.80:
61.222.158.229.80 170.129.130.226.80:
61.222.177.125.80 170.129.130.226.80:
61.222.177.133.80 170.129.130.226.80:
61.222.177.141.80 170.129.130.226.80:
163.22.229.253.80 170.129.31.29.80:
163.23.238.9.80 170.129.44.252.80:
192.192.171.251.80 170.129.192.213.80:
192.192.90.201.80 170.129.130.226.80:
202.29.28.1.80 170.129.139.116.80:
202.29.28.1.80 170.129.185.21.80:
202.29.28.1.80 170.129.31.152.80:
```

This command shows that a variety of external and internal hosts are involved in the traffic. The traffic is seen only from the outside hosts to the inside hosts. Since the 202.x.x.x/8 Class A has also involved in some other scanning activities previously in the logs file, we will need to verify is those IPs are not part of a broader attack in our internal segment.

Bellow is a sample of the packets used for the attack:

```
$ tcpdump -ttt -nvvv -r 2002.10.17 "tcp[0:2] == tcp[2:2]"
000000 IP (tos 0x0, ttl 44, id 56514, len 40) 202.29.28.1.80 > 170.129.185.21.80: . [tcp sum ok]
178:178(0) ack 0 win 1400
10. 000000 IP (tos 0x0, ttl 44, id 56770, len 40) 202.29.28.1.80 > 170.129.185.21.80: . [tcp sum
ok] 111:111(0) ack 1 win 1400
10. 010000 IP (tos 0x0, ttl 44, id 57025, len 40) 202.29.28.1.80 > 170.129.185.21.80: . [tcp sum
ok] 227:227(0) ack 1 win 1400
10. 000000 IP (tos 0x0, ttl 44, id 57272, len 40) 202.29.28.1.80 > 170.129.185.21.80: . [tcp sum
ok] 333:333(0) ack 1 win 1400
10. 000000 IP (tos 0x0, ttl 44, id 57504, len 40) 202.29.28.1.80 > 170.129.185.21.80: . [tcp sum
ok] 433:433(0) ack 1 win 1400
8008. 970000 IP (tos 0x0, ttl 44, id 36247, len 40) 202.29.28.1.80 > 170.129.31.152.80: . [tcp sum
ok] 510:510(0) ack 0 win 1400
9. 990000 IP (tos 0x0, ttl 44, id 36504, len 40) 202.29.28.1.80 > 170.129.31.152.80: . [tcp sum
ok] 110:110(0) ack 1 win 1400
10. 000000 IP (tos 0x0, ttl 44, id 36802, len 40) 202.29.28.1.80 > 170.129.31.152.80: . [tcp sum
ok] 236:236(0) ack 1 win 1400
10. 040000 IP (tos 0x0, ttl 44, id 37072, len 40) 202.29.28.1.80 > 170.129.31.152.80: . [tcp sum
ok] 353:353(0) ack 1 win 1400
```

2. Detect was generated by:

Detect was generated by snort according to the source of data. The packets recorded in the file are only packets that violated the source signature file. The following signature have generated some of those alarms:

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap TCP"; flags:A,12; ack:0; reference:arachnids,28; classtype:attempted-recon; sid:628; rev
:2;)

We noticed in the traces than some of the packets have an ACK number equal to 1. The IDS might have a similar rule for traffic with and ack equal to 1 or any traffic with a destination port lower than 1024 like:

alert tcp $EXTERNAL_NET any -> $HOME_NET :1024 (msg:"Low port traffic"; classtype:attempted-recon;)

We used tcpdump and ethereal to perform the analysis.

4. Description of the attack:
The attack consists in trying to gather information on the network remote hosts using loose ACK packets. ACK scan can reveal whether or not a host is behind a firewall or packet filtering device. A packet filtering device rejecting the packet will drop the packet or send an "ICMP host unreachable" or passes it to the server. Most of the servers would then reply with a RST packet since no connection is established.

4. Probability the source address was spoofed:
Even though I do not have a complete trace of the network activity (only the packet that matched the rules), the packets seen do not seem to be part of an established session mainly because the ACK numbers are set to either 0 or 1 in most of the cases which doesn't make any sense, all the sequence numbers are smaller than 2000 (for a 32bit field) and all the TTLs are between 43 – 49. The packets have all the signs of being crafted therefore the packet IP address may very well have been spoofed.

5. Attack mechanism:
The following frame has been extracted from the logs. In green are the common fields shared by all the frames.

```
Frame 544 (60 bytes on wire, 60 bytes captured)
    Arrival Time: Nov 17, 2002 10:55:03.666507000
    Packet Length: 60 bytes
    Capture Length: 60 bytes
Ethernet II, Src: 00:03:e3:d9:26:c0, Dst: 00:00:0c:04:b2:33
    Type: IP (0x0800)
    Trailer: 000000000000q
Internet Protocol, Src Addr: 61.218.15.126, Dst Addr: 170.129.192.213
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 40
    Identification: 0x7442 (29762)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 49
    Protocol: TCP (0x06)
    Header checksum: 0x5cdf (correct)
Transmission Control Protocol, Src Port: http (80), Dst Port: http (80), Seq: 1015, Ack: 0, Len: 0
    Flags: 0x0010 (ACK)
    Window size: 1400
    Checksum: 0xed16 (correct)
```

The fields tagged in green are the most remarkable fields in this packet:

- small packet size: 60bytes
- rather small TTL (49 here)
- reflective port numbers (80/TCP)
- low sequence number
- TCP/ACK packet
- Small window size: 1400

The following commands confirm the patterns seen:

```
$ tcpdump -nvr 2002.10.17 "tcp[0:2] == tcp[2:2] and tcp[4:4] < 0x7D0" | wc -l
36
```

The reflective port packet all have a sequence number smaller than 2000.

- **Packet size**

```
$ tcpdump -nvr 2002.10.17 "tcp[0:2] == tcp[2:2] and tcp[4:4] < 0x7D0" | cut –d ' ' -f 10 | uniq -c
     36 40)
```

All the packets in the attack have the same size

- **TTL**

The first value represents the number of occurrence and the second number the associated TTL value. (format is “number of hit”, “TTL value,” e.g. first line: 5 occurrences with a ttl value of 43).

```
$ tcpdump -nvr 2002.10.17 "tcp[0:2] == tcp[2:2] and tcp[4:4] < 0x7D0" | cut –d ' ' -f 6 | sort –n |
uniq -c
      5 43,
     16 44,
      4 47,
      7 48,
      4 49,
```

All the packets have a TTL is between 43 and 49.

- **Port used**

(Each line represents the number of occurrence between two ports: here 36 reflective port packets went from port 80 to port 80)

```
$ tcpdump -nvr 2002.10.17 "tcp[0:2] == tcp[2:2] and tcp[4:4] < 0x7D0" | cut -d' ' -f11,13 | cut -d.
-f5,9 | wc –l
     36 80 80:
```

All the packets are coming from port 80 going to port 80.

- **Flags**

```
$ tcpdump -nvr 2002.10.17 "tcp[0:2] == tcp[2:2] and tcp[4:4] < 0x7D0" | cut –d ' ' -f 18 | uniq -c
     36 ack
```

All the packets are ACK packets

- **Window size**

```
$ tcpdump -nvr 2002.10.17 "tcp[0:2] == tcp[2:2] and tcp[4:4] < 0x7D0" | cut –d ' ' -f 21 | uniq -c
     36 1400
```

All the packets have a window size of 1400.

To summarize the attack can be identified as a series of:

- reflective port ACK packet
- port and destination set to 80
- TTL set to a value between 43 and 49
- A window size set to 1400

An analysis of the timestamps gives additional clues about the pattern seen. I dumped with tcpdump the 36 packets from the source file in EPOCH time (option –tt) and used excel to calculate the time difference between each packets:

Epoch time	Source IP	Destination IP	Delta t in s
1037493547	202.29.28.1.80	170.129.185.21.80:	

1037493557	202.29.28.1.80	170.129.185.21.80:	10
1037493567	202.29.28.1.80	170.129.185.21.80:	10
1037493577	202.29.28.1.80	170.129.185.21.80:	10
1037493587	202.29.28.1.80	170.129.185.21.80:	10
1037501596	202.29.28.1.80	170.129.31.152.80:	8009
1037501606	202.29.28.1.80	170.129.31.152.80:	10
1037501616	202.29.28.1.80	170.129.31.152.80:	10
1037501626	202.29.28.1.80	170.129.31.152.80:	10
1037529755	61.221.99.242.80	170.129.31.29.80:	28129
1037529760	61.221.99.242.80	170.129.31.29.80:	5
1037529766	163.22.229.253.80	170.129.31.29.80:	6
1037530237	61.218.161.202.80	170.129.44.252.80:	471
1037530252	61.218.161.210.80	170.129.44.252.80:	15
1037530260	163.23.238.9.80	170.129.44.252.80:	8
1037559299	61.218.15.118.80	170.129.192.213.80:	29039
1037559304	61.218.15.126.80	170.129.192.213.80:	5
1037559314	61.221.88.198.80	170.129.192.213.80:	10
1037559319	61.221.88.198.80	170.129.192.213.80:	5
1037559327	192.192.171.251.80	170.129.192.213.80	8
1037559332	192.192.171.251.80	170.129.192.213.80	5
1037565176	202.29.28.1.80	170.129.139.116.80:	5844
1037565186	202.29.28.1.80	170.129.139.116.80:	10
1037565196	202.29.28.1.80	170.129.139.116.80:	10
1037565216	202.29.28.1.80	170.129.139.116.80:	20
1037565226	202.29.28.1.80	170.129.139.116.80:	10
1037565236	202.29.28.1.80	170.129.139.116.80:	10
1037568708	61.222.154.109.80	170.129.130.226.80:	3473
1037568713	61.222.158.229.80	170.129.130.226.80:	5
1037568718	61.222.158.229.80	170.129.130.226.80:	5
1037568729	61.222.177.125.80	170.129.130.226.80:	11
1037568735	61.222.177.133.80	170.129.130.226.80:	6
1037568740	61.222.177.133.80	170.129.130.226.80:	5
1037568750	61.222.177.141.80	170.129.130.226.80:	10
1037568758	192.192.90.201.80	170.129.130.226.80:	7
1037568763	192.192.90.201.80	170.129.130.226.80:	5

The time between each scan seems to be constant over the time.

I found in the incidents mailing lists logs with similar characteristics and gad between the requests:
http://www.incidents.org/archives/intrusions/msg08110.html

Those traces that looks like an evil attack are actually generated by a server load balancer from Radware© called linkprook as a part of an IP proximity algorithm. The load balancer uses active probing to determine the best path between two systems. The Radware patent application for their proximity algorithm details the process (patent number 115643):
http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=/netahtml/search-bool.html&r=2&f=G&l=50&co1=AND&d=ptxt&s1=radware&OS=radware&RS=radware

The patent description explains:

> *"Sending a TCP ACK message to client 26 may be used where pinging would otherwise fail due to an intervening firewall or NAT device filtering out a polling message […] A TCP ACK may be sent to the client's source IP address and port. If the client's request was via a UDP connection, a TCP ACK to the client's source IP address and port 80 may be used. One or both TCP ACK messages should bypass any intervening NAT or firewall and cause client 26 to send a TCP RST message, which may be used to determine both latency and TTL"*

The patent also mentions the use of "other technique" in case the first polling method one fails. We found 3 other suspicious packets with the same TTL and window size than the packet analyzed above which might as well have been generated by the load balancer.

```
$ tcpdump -nvr 2002.10.17 "host 170.129.50.122"
03:43:17.236507 IP (tos 0x0, ttl 44, id 33637, len 40) 202.28.21.2.80 > 170.129.50.122.53: .
[tcp sum ok] ack 0 win 1400
03:43:17.546507 IP (tos 0x0, ttl 42, id 33656, len 40) 203.146.247.2.80 > 170.129.50.122.53: .
[tcp sum ok] ack 0 win 1400
03:43:17.576507 IP (tos 0x0, ttl 42, id 33657, len 40) 203.155.14.2.81 > 170.129.50.122.53: .
[tcp sum ok] ack 0 win 1400
```

Finally, the patent also mentioned using triangulation between 2 or more load balancers to determine IP proximity, explaining the distributed pattern seen above. The probe targeting 170.129.130.226 (see timestamp table above) demonstrates the apparent distribution of the scan caused by *radware* triangulation algorithm. Each probe query the .226 address several times at a 5s interval and each probes are separated by a 10s interval. We can also clearly see in the table above that the time between the queries of different destination address is random between each series of probes. Unfortunately the pattern did not mention anything about the timing between the requests and the different pollers involved in the triangulation. However, some logs sent in the incidents.org mailing list show similar gaps between the TCP ACK packets (5 or 6s):
http://www.incidents.org/archives/intrusions/msg08110.html

Note that the IDS might not see the packets sent to the domain primary DNS (as mentioned by Chris Brenton) because the servers might be located on a DMZ not monitored by the IDS.

Note that the IDS might not see the packets sent to the domain primary DNS (as mentioned by Chris Brenton) because the servers might be located on a DMZ not monitored by the IDS.

6. Correlation

A report form the global incident analysis center written by Matt Fearnow describes the issue:

http://www.sans.org/y2k/040301-1430.htm

The article of Chris Brenton helped us to interpret those results:
http://www.incidents.org/archives/intrusions/msg08129.html

The Radware patent application for their proximity algorithm gives more information on the algorithm used:
http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=/netahtml/search-bool.html&r=2&f=G&l=50&co1=AND&d=ptxt&s1=radware&OS=radware&RS=radware

Some logs sent in the incidents.org mailing list show similar gaps between the TCP ACK packets (5 or 6s):
http://www.incidents.org/archives/intrusions/msg08110.html

7. Evidence of active targeting:
The packets seen are the result of a request made by a user to a server load balanced by a Radware linkproof device. The Radware is probing the network only after a request was received by the load balancer.

8. Severity
S = (5 + 2) – (2+2) = 3

Critically: The packets, even if they are coming from a load balancer, are probing the internal network, critical for the organization, gathering critical information on the organization network.

Lethality: The attack by itself is not lethal. The main goal of those reconnaissance packets is to scan the network. However, this scan can lead to a devastating attack on the network by revealing the intrinsic security architecture.

System countermeasures: A system firewall configured in stealth mode (drop TCP packets, do not send any reply like RST) would drop the TCP traffic seen. The administrator could also use an IP filter and drop any packets with an ACK set to 0 or 1, a window of 1400 and a sequence number smaller than 2000.

Network countermeasures: a simple stateful Firewall configured to drop non stateful traffic would certainly prevent any network scanning activity (hostile or friendly).

9. Defensive recommendation
A stateful firewall could be added (If it is not already the case) to protect the internal clients from this kind of probes. Most of the stateful firewall can be configured in stealth mode to prevent them to send a RST packet on a loose ACK packet.

10. Multiple choice test question

Radware Linkproof load balancers are known to determine the round trip time and latency between a subnet and one of its servers by actively probing with a different combination of packets the network subnet to get some type of reply (ICMP reply, RST…). Which one of those packets is more likely to come from a linkproof loadbalancer?

a. 14:31:53.306507 IP 10.10.10.10.80 > 20.20.20.20.80: . ack 0 win 1400
b. 14:31:53.306507 IP 10.10.10.10.80 > 20.20.20.20.61385: . ack 14350 win 24000
c. 14:31:53.306507 IP 10.10.10.10.80 > 20.20.20.20.8080: . ack 3666 win 65535
d. 14:31:53.306507 IP 10.10.10.10.80 > 20.20.20.20.1516: . 55481:56941(1460) ack 855 win 65535 (DF)

Answer: a

Because the packet A is looking for a RST or ICMP host unreachable packet from the local packet filtering device using a typical ACK scan pattern. The rest of the packets to not have a suspicious pattern.

Question/Comment:
I did not receive any question on this analysis.

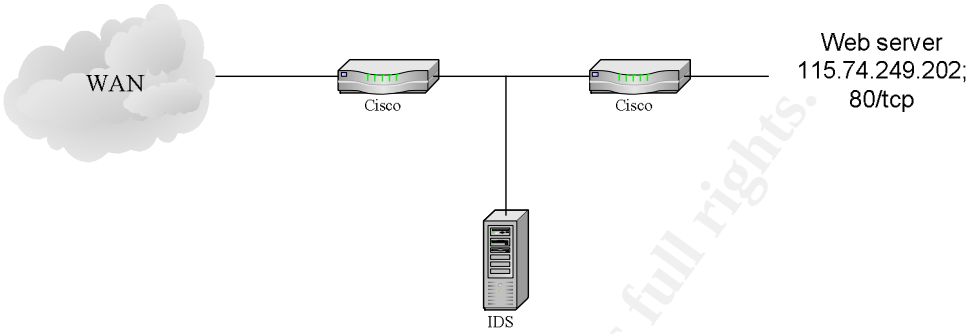
Attack #3: 2002.8.28 – Spam relay scanning

1. Source of trace: incidents.org URL: www.incidents.org/logs/Raw/2002.8.28

This analysis focuses on a number of packets with an HTTP request attempt with some pretty strange path; the rest of the logs will tell us if the attack was coordinated, distributed or localized to a single device. First, let's try to determine where the IDS is located on the network. The following command will help us determine the MAC address on each side of the IDS:

```
> Tcpdump -neqr 2002.8.28 | cut -d ' ' -f2,3 | sort –n | uniq
    00:00:0c:04:b2:33 00:03:e3:d9:26:c0
    00:03:e3:d9:26:c0 00:00:0c:04:b2:33
```

The MAC addresses are the same than for the detect #1. Both OUI (Organizationally Unique Identifier: first three-octets of the MAC address) are assigned to CISCO. Based on those MAC the IDS is tapped between two CISCO routers 00:03:e3 being the uplink. 00:00:0c would then be the next hop to the internal segment 115.x.x.x/8. (see diagram bellow).



The target of the attack is an internal host: 115.74.249.202. You will find bellow a sample of the packets with their payload, involving the targeted host IP address dumped using ethereal for each hosts involved in the web scan:

```
Frame 143 (515 bytes on wire, 515 bytes captured)
Ethernet II, Src: 00:03:e3:d9:26:c0, Dst: 00:00:0c:04:b2:33
Internet Protocol, Src Addr: 24.189.224.108 (24.189.224.108), Dst Addr: 115.74.249.202
(115.74.249.202)
Transmission Control Protocol, Src Port: 2952 (2952), Dst Port: http (80), Seq: 538350292, Ack:
2777298622, Len: 461
Hypertext Transfer Protocol
    GET /cgi-
bin/FormMail.pl?recipient=<cgiscripts@ziplip.com>nobody@XXXXXXXX&subject=http://www.XXXXXXXX/cgi-
bin/FormMail.pl/&email=John@doe.com&\240=is anybody out there? HTTP/1.1\r\n
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*\r\n
    Accept-Language: en-us\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Accept-Encoding: gzip, deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)\r\n
    Host: www.XXXXXXXX\r\n
    Connection: Keep-Alive\r\n
    \r\n
Frame 144 (517 bytes on wire, 517 bytes captured)
Ethernet II, Src: 00:03:e3:d9:26:c0, Dst: 00:00:0c:04:b2:33
Internet Protocol, Src Addr: 24.189.224.108 (24.189.224.108), Dst Addr: 115.74.249.202
(115.74.249.202)
Transmission Control Protocol, Src Port: 2953 (2953), Dst Port: http (80), Seq: 538411984, Ack:
2773922596, Len: 463
Hypertext Transfer Protocol
    GET /cgi-
bin/FormMail.cgi?recipient=<cgiscripts@ziplip.com>nobody@XXXXXXXX&subject=http://www.XXXXXXXX/cgi-
bin/FormMail.cgi/&email=John@doe.com&\240=is anybody out there? HTTP/1.1\r\n
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*\r\n
```

The frames number 109 and 111 bellow actually contains a HTTP 403 reply from the internal host (frame bellow) which indicates that 115.74.249.202 has actually a web server running. According to the HTTP tags, the web server is running RedHat Linux, Apache

1.3.12 as well as Frontpage 4.0.4.3. A quick analysis of the packet TTL, window, DF and TOS field confirm the OS type. 115.74.249.202 is therefore a server open to the internet on port 80/TCP.

```
Frame 109 (590 bytes on wire, 590 bytes captured)
Ethernet II, Src: 00:00:0c:04:b2:33, Dst: 00:03:e3:d9:26:c0
Internet Protocol, Src Addr: 115.74.249.202 (115.74.249.202), Dst Addr: 195.29.132.167
(195.29.132.167)
Transmission Control Protocol, Src Port: http (80), Dst Port: 1425 (1425), Seq: 2728099470, Ack:
5685133, Len: 536
Hypertext Transfer Protocol
    HTTP/1.1 403 Forbidden\r\n
    Date: Sat, 28 Sep 2002 14:39:20 GMT\r\n
    Server: Apache/1.3.12 (Unix)  (Red Hat/Linux) FrontPage/4.0.4.3\r\n
    Keep-Alive: timeout=15, max=100\r\n
    Connection: Keep-Alive\r\n
    Transfer-Encoding: chunked\r\n
    Content-Type: text/html; charset=iso-8859-1\r\n
    X-Pad: avoid browser bug\r\n
    \r\n
           Data (252 bytes)
```

2. The detect was generated by:
According to the README file located in the log directory, the logs were generated from a Snort NIDS.

I tried to run snort with the standard signature file v1.124
"snort –d -r 2002.8.28 –l logs -c rules\snort.conf"

However snort did not return any alarm because the full logs would be require to be able to reconstruct the TCP data flow to run the Web-IIS signature file.

I "grepped" on the default snort signature file (v1.124) and found a match in the WEB-CGI signature:

> alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-CGI formmail access"; flow:to_server,established; uricontent:"/formmail"; nocase; reference:nessus,10782; reference:nessus,10076; reference:bugtraq,1187; reference:cve,CVE-1999-0172; reference:arachnids,226; classtype:web-application-activity; sid:884; rev:8;)

in other words snort will alarms here if the string /formmail is in the URL for an established session to a server. The signature entry includes references to security and vulnerability database:

http://cgi.nessus.org/plugins/dump.php3?id=10782
http://cgi.nessus.org/plugins/dump.php3?id=10076
http://online.securityfocus.com/bid/1187
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0172
http://www.whitehats.com/info/IDS226/

Note that even if the server replied with a HTTP 404 error (page not found) referencing the formmail attempt, the server reply will not show in the logs because the signature specifies “to_server”.

3. Probability the source address was spoofed

Since 115.74.249.202 is a web server and that HTTP requests can be sent after a TCP 3 way handshake occurred, the connection between the client and the server already occurred and therefore makes spoofing difficult. The attacker may however have disguised his identity using an HTTP proxy or a SOCK server.

The source address (24.189.224.108) resolves as *ool-18bde06c.dyn.optonline.net* and belong according to ARIN to “Optimum Online (Cablevision Systems)” a cable modem provider. Bellow is an extract of the ARIN whois entry:

```
CustName:   Optimum Online (Cablevision Systems)
Address:    111 New South Road
City:       Hicksville
StateProv:  NY
PostalCode: 11801
Country:    US
RegDate:    2003-05-30
Updated:    2003-05-30
NetRange:   24.189.224.0 - 24.189.239.255
CIDR:       24.189.224.0/20
NetName:    OOL-CBCORMNY1-0821
NetHandle:  NET-24-189-224-0-1
Parent:     NET-24-188-0-0-1
NetType:    Reassigned
Comment:
RegDate:    2003-05-30
       Updated:    2003-05-30
```

Cable modem sources IPs are very frequently used in attacks. Most of the cable users are using poorly protected system with known vulnerabilities making them an easy target for scripts kiddies.

4. Description of the attack:

The attack is a stimulus: the attacker is actively probing a remote system to determine its characteristics. The traces seen above are scan directed against an internet web server. The attacker is probing a remote server with an unprotected script named formmail.pl or formmail.cgi.

5. Attack mechanism:

There is two well known formmail attack:

- The first one attempts to locate a formmail cgi to execute command on the server. According to the bugtraq description (http://www.securityfocus.com/bid/1187/discussion/):

"This is accomplished by specifying a particular CGI environmental variable such as PATH, DOCUMENT_ROOT, SERVER_PORT in the specially formed URL which will email the results to the address given"

None of the formmail queries seen above actually contains those environmental variables.

- The second one tries to locate formmail.pl to send unsolicited emails from the server as described by Michael Palamar in securityteam.com: http://www.securiteam.com/securitynews/Formmail_pl_Can_Be_Used_As_An_Open_Mail_Relay.html
  As explained in the article, the formmail.pl check the HTTP_REFERER value on the web server come from a accepted domain. The attacker gets around that problem by spoofing the HTTP_REFERER in a crafted HTTP request. In our case the URL will come with an empty HTTP_REFERER value. If the server is not configured properly with a valid list of referrer, the server will execute formmail and send the email.

  The email is sent to "nobody@XXXXXXXX" from 'John@doe.com' with the following subject: http://www.XXXXXXXX/cgi-bin/FormMail.pl/: with www.XXXXXXX the server scanned and "is anybody out there" in the body. (pattern conformed by the "formmail hall of shame" entry mentioned in the sub-section 6).

  The attack happens in several steps:
  1. First the evil users setup a valid mailbox on a free email services (such as ziplip.com, hushmail.com, yahoo.com etc…)
  2. the evil user a large list of web servers (the list can be gathered from a search engine such as google)
  3. The evil user run a script with the list of web servers on formmail typical locations (/cgi, /cgi-bin…) attempting blindly to run the script with a inexistent reply email address (parameter *email* of formmail), the email address created in the first step as the email destination (parameter *recipient* of formmail) and the URL to the formmail script as the subject (parameter *title* of formmail). Text in the body is optional.
  4. the Evil attacker check the mailbox created in the first step (more than likely using a public proxy). Having the URL directly in the subject allows him/her to copy and paste directly from the interface without having to open the email.

As I looked at the traffic to the web server, I saw more attempts to locate a formmail as well as other scanning activities against the web server. I do not believe that the formmail scans are related to the attack for the following reasons:

- Formmail is queried several times from different IPs. If the scan was coordinated, the attacker would no redo the same test

- By looking at the frames bellow we can see that the mandatory "0d 0a" after "HTTP/1.1" and the request header (in that case "content-type"). The logs I found from "waldo kitty" in Spamcops (http://news.spamcop.net/pipermail/spamcop-list/2002-April/000254.html) and the analysis from Carl Gibbons

(http://www.du.edu/~cgibbons/GCIA-attempt/2002-10-15formmail.html) highlighted the same anomaly which lead us to think that this pattern would result from a bug in the client. Since the formmail frame I analyzed above did not have that anomaly we can definitely say that those scans are not related.

```
>tcpdump -nX -r 2002.8.28 "host 4.63.173.119"
19:56:58.246507  IP  4.63.173.119.3863  >  115.74.249.202.80:  P  3455820667:3455820981(314)  ack
2881029548 win 15000 (DF)
0x0000   4500 0162 23cc 4000 6d06 5a8f 043f ad77        E..b#.@.m.Z..?.w
0x0010   734a f9ca 0f17 0050 cdfb a37b abb9 05ac        sJ.....P...{....
0x0020   5018 3a98 e2fc 0000 4745 5420 2f63 6769        P.:.....GET./cgi
0x0030   2d62 696e 2f66 6f72 6d6d 6169 6c2e 706c        -bin/formmail.pl
0x0040   3f65 6d61 696c 3d66 3240 616f 6c2e 636f        ?email=f2@aol.co
0x0050   6d26 7375 626a 6563 743d 7777 772e 5858        m&subject=www.XX
0x0060   5858 5858 5858 2f63 6769 2d62 696e 2f66        XXXXXX/cgi-bin/f
0x0070   6f72 6d6d 6169 6c2e 706c 2672 6563 6970        ormmail.pl&recip
0x0080   6965 6e74 3d70 6869 7368 7461 6e6b 5f30        ient=phishtank_0
0x0090   3030 3240 7961 686f 6f2e 636f 6d26 6d73        002@yahoo.com&ms
0x00a0   673d 7730 3074 2030 7961 686f 6f25 3245        g=w00t.0yahoo%2E
0x00b0   636f 6d26 6d73 673d 7730 3074 2048 5454        com&msg=w00t.HTT
0x00c0   502f 312e 3143 6f6e 7465 6e74 2d54 7970        P/1.1Content-Typ
0x00d0   653a 2061 7070 6c69 6361 7469 6f6e 2f78        e:.application/x
0x00e0   2d77 7777 2d66 6f72 6d2d 7572 6c65 6e63        -www-form-urlenc
0x00f0   6f64 6564 0d0a 5573 6572 2d41 6765 6e74        oded..User-Agent
0x0100   3a20 476f 7a69 6c6c 612f 342e 3020 2863        :.Gozilla/4.0.(c
0x0110   6f6d 7061 7469 626c 653b 204d 5349 4520        ompatible;.MSIE.
0x0120   352e 353b 2077 696e 646f 7773 2032 3030        5.5;.windows.200
0x0130   3029 0d0a 486f 7374 3a20 7777 772e 5858        0)..Host:.www.XX
0x0140   5858 5858 5858 0d0a 436f 6e6e 6563 7469        XXXXXX..Connecti
0x0150   6f6e 3a20 4b65 6570 2d41 6c69 7665 0d0a        on:.Keep-Alive..
0x0160   0d0a                                           ..

>tcpdump -nX -r 2002.8.28 -nX -r 2002.8.28 "host 63.16.15.140"
16:22:53.186507 IP 63.16.15.140.2199 > 115.74.249.202.80: P 81017162:81017482(320) ack 3230826048
win 9520 (DF)
0x0000   4500 0168 bec9 4000 7306 1ca6 3f10 0f8c        E..h..@.s...?...
0x0010   734a f9ca 0897 0050 04d4 394a c092 7e40        sJ.....P..9J..~@
0x0020   5018 2530 a13b 0000 4745 5420 2f63 6769        P.%0.;..GET./cgi
0x0030   2d62 696e 2f46 6f72 6d4d 6169 6c2e 706c        -bin/FormMail.pl
0x0040   3f65 6d61 696c 3d53 6b61 6e6e 6564 4061        ?email=Skanned@a
0x0050   6f6c 2e63 6f6d 2673 7562 6a65 6374 3d77        ol.com&subject=w
0x0060   7777 2e58 5858 5858 5858 582f 6367 692d        ww.XXXXXXXX/cgi-
0x0070   6269 6e2f 466f 726d 4d61 696c 2e70 6c26        bin/FormMail.pl&
0x0080   7265 6369 7069 656e 743d 6368 6f6b 6f73        recipient=chokos
0x0090   7973 3431 3340 686f 746d 6169 6c2e 636f        ys413@hotmail.co
0x00a0   6d26 6d73 673d 6d69 536c 6564 544d 2025        m&msg=miSledTM.%
0x00b0   3245 636f 6d26 6d73 673d 6d69 536c 6564        2Ecom&msg=miSled
0x00c0   544d 2048 5454 502f 312e 3143 6f6e 7465        TM.HTTP/1.1Conte
0x00d0   6e74 2d54 7970 653a 2061 7070 6c69 6361        nt-Type:.applica
0x00e0   7469 6f6e 2f78 2d77 7777 2d66 6f72 6d2d        tion/x-www-form-
0x00f0   7572 6c65 6e63 6f64 6564 0d0a 5573 6572        urlencoded..User
0x0100   2d41 6765 6e74 3a20 476f 7a69 6c6c 612f        -Agent:.Gozilla/
0x0110   342e 3020 2863 6f6d 7061 7469 626c 653b        4.0.(compatible;
0x0120   204d 5349 4520 352e 353b 2077 696e 646f        .MSIE.5.5;.windo
0x0130   7773 2032 3030 3029 0d0a 486f 7374 3a20        ws.2000)..Host:.
0x0140   7777 772e 5858 5858 5858 5858 0d0a 436f        www.XXXXXXXX..Co
0x0150   6e6e 6563 7469 6f6e 3a20 4b65 6570 2d41        nnection:.Keep-A
       0x0160   6c69 7665 0d0a 0d0a                            live....
```

As for the rest attacks targeting the server, I could list to three categories of attacks:

- directory traversal attempts
- repeated FrontPage extension scans (_vti_inf POST attempts) from a unique host
- IIS propfind attack

Those attacks are all coming from different IPs but all of them contain a 'via' or "X-cache-ID" tag in the request indicating that the client used an HTTP proxy to connect to the server. The same attacks are reiterated at various points in time throughout the logs (but from different source IP) which indicates that the attack is not coordinated.

6. Correlation

Les M Gordon analyzed a Formmail scan in his GCIA assignment coming from difference source than the one I analyzed here:
http://www.sans.org/rr/papers/23/1096.pdf
The methods used in the attack he analyzed where not as advanced as the one described above since the script was scanning only for the existence of the formmail script.

The host used as well as the email used in the attack have been reported to the "formmail hall of shame" attempting to scan www.softwolves.pp.se/cgi-bin/formmail.pl from 67.194.4.28 and 24.189.224.108:
http://www.softwolves.pp.se/misc/formmail_hall_of_shame/0208

7. Evidence of active targeting:

The attack is a scan, there is no evidence from the logs I have analyzed that this particular server was targeted. Actually, this server is probably part of a larger scan since I found several entries in the "formmail hall of shame" for the exact same pattern seen in our analysis.

8. Severity

S = (3 + 2) – (2 + 2) = 1

Critically: The system targeted might be critical to the targeted organization.

Lethality: Beyond a simple SPAM attempt, the formmail attack could reveal critical information on the system.

System countermeasures: Formmail would need to be kept up to date. The administrator could request to have a static string set on the e-mail title to have sendmail detect and filter the unsolicited emails.

Network countermeasures: a reverse proxy or application firewall (Sanctum Appshield) can be used to filter the requests to the web server.

9. Defensive recommendation

Of course I would strongly recommend maintaining the server updated with the latest patch version and recommend configuring the web server to not display the version of each module on an error would prevent leaking of crucial information such as the one I have seen in Frame 109.

I would also recommend, if the server is critical for the organization, to use an Application firewall (like Sanctum Appshield) to control the fields used to call the CGI.

10. Multiple choice test question
What following HTTP tag is definitely indicating that an HTTP client is definitely using an HTTP proxy?
a. “User-Agent: Microsoft-WebDAV-MiniRedir/5.1.2600”
b. “Cache-Control: max-age=172800”
c. “Via: 1.1 teradant8:8080 (Squid/2.3.STABLE4)\r\n”
d. “Cookie: noproxy=1”

Answer = c

Via: is added by proxy server to indicates that an HTTP request has been replayed by a proxy server.

Questions/comments:
No question has been asked on this analysis.

PART 3: ANALYZE THIS

Security audit report

1. Executive summary of the analysis

Between July 28th and august first 2003 the university intrusion detection system generated 360,949 alerts, 4,850,111 scans and 271962 out of specification packets. We focused in this analysis on the university internal systems integrity, dangerous internal activity and IDS signature file configuration. This analysis will provide a list of internal systems to be assessed and external systems to keep an eye on, as well as recommendations to improve the IDS performances. Due to the high volume of data to assess, the analysis will focus on the most critical alarms and the ones occurring the most often.

According the activity we have been observing through the logs, we would strongly recommend the following systems to be audited for trojans:

MY.NET.198.221
MY.NET.74.216
MY.NET.3.56
MY.NET.3.54

We would also recommend checking the following systems for potential virus or Trojans:

MY.NET.137.7
MY.NET.30.4
MY.NET.30.3

We noticed that even if numerous hosts were involved in the alarms, a great number of attacks were initiated from a limited number of providers. We would recommend opening cases with the owner of the following IP addresses:

216.95.201.0
193.252.203.96
81.48.143.73
66.82.245.45

Finally, the analysis of the logs revealed several weaknesses in the university architecture. I would recommend the university to tighten the firewall (or any access control device) to allow external traffic (traffic coming from the internet) only to a limited number of server and ports. The IDS should be configured with rules closer to standard rules to

minimize the amount of false positive and optimize the detection process. I would also recommend to put in place (if it is not already the case) a security policy regarding the use of internet resources to download copyrighted materials. I would also restrict the access to internal documents such as equipment list, equipment configuration, and any internal document that would reveal network topology. We would remove the hardware equipment list found in http://www.gl.umbc.edu/hardware.shtml as well as any other sensitive material.

2. The files

We choose the logs created between July 28th 20003 and August 1st 2003.

Alert Files	Size	Scan Files	Size	Out-Of-Spec Files*	Size
				OOS_Report_2003_07_28_29050	952323
alert.030728.gz	19761665	scans.030728.gz	75178701	OOS_Report_2003_07_29_23718	4418563
alert.030729.gz	19049691	scans.030729.gz	66558283	OOS_Report_2003_07_30_29913	1274883
alert.030730.gz	16531539	scans.030730.gz	62857037	OOS_Report_2003_07_31_11092	1469443
alert.030731.gz	15892552	scans.030731.gz	43870208	OOS_Report_2003_08_01_5880.	2585603
alert.030801.gz	21854256	scans.030801.gz	53788058	OOS_Report_2003_08_02_26778	1208323

* note that the date used for the out of spec (OOS) files name were actually offset by 24h. For example the "OOS_Report_2003_07_28_29050" list OOS packets received July 27th.

Those logs where all generated during the working days (Monday to Friday) of the week or the 27th of July 2003.

3. Relationship between the different hosts

The following table list the internal hosts as well as the services used based on the traffic seen in the logs. We determined the services available based first on the outbound traffic source ports (<1024) seen in the alert file for the five days analyzed. Since those systems sent packets from those services source port, there is a high probability that the system is actually listening on that port too. We added in parenthesis the name of service commonly used for the related port number. We also used DNS to determine the main function of the server.

MY.NET.5.20	80(HTTP)
MY.NET.5.44	80(HTTP)
MY.NET.6.7	80(HTTP)
MY.NET.12.4	10(POP3)
MY.NET.12.4	143(IMAP)
MY.NET.12.4	993(IMAPS)
MY.NET.12.6	25(SMTP)
MY.NET.24.20	69(TFTP)
MY.NET.24.34	69(TFTP)
MY.NET.24.34	80(HTTP)

MY.NET.24.44	80(HTTP)
MY.NET.24.58	443(HTTPS)
MY.NET.25.72	25(SMTP)
MY.NET.25.73	25(SMTP)
MY.NET.29.11	443(HTTPS), 80(HTTP)
MY.NET.29.66	80(HTTP)
MY.NET.30.3	80(HTTP)09,524(Novell ncp)
MY.NET.30.4	80(HTTP)09,51443,524(ncp),80(HTTP)
MY.NET.32.167	80(HTTP)
MY.NET.60.14	80(HTTP)
MY.NET.69.217	80(HTTP)
MY.NET.70.185	80(HTTP)
MY.NET.84.216	3589(ChiliASP/isomair)
MY.NET.100.165	80(HTTP), 21(FTP)
MY.NET.115.10	80(HTTP)
MY.NET.150.83	80(HTTP)
MY.NET.198.221	69(TFTP)

Next we determined the different subnets functions based on the IP address present in the logs and a publicly available comprehensive list of hosts we found on one of the HTTP servers listed above (MY.NET.60.14):
http://www.gl.umbc.edu/hardware.shtml

MY.NET.5.0	Monitoring – system support
MY.NET.6.0	Web Servers – application servers
MY.NET.12.0	Mail servers
MY.NET.24.0	Internal servers and network management
MY.NET.25.0	?
MY.NET.29.0	University groups web sites
MY.NET.30.0	Novell NetWare support
MY.NET.32.0	“Peoplesoft” servers (finance department)
MY.NET.53.0	User network access
MY.NET.60.0	University computing services
MY.NET.69.0	User network access
MY.NET.70.0	UCS
MY.NET.72.0	User network access
MY.NET.73.0	User network access
MY.NET.74.0	User network access
MY.NET.75.0	CHPDM network
MY.NET.80.0	User network access
MY.NET.83.0	User network access
MY.NET.84.0	User network access
MY.NET.97.0	Dial-up access
MY.NET.98.0	Dial-up access
MY.NET.100.0	Computer Science network

MY.NET.115.0	Biotech network
MY.NET.150.0	Library network(?)
MY.NET.163.0	User network access (physic lab?)
MY.NET.178.0	User network access
MY.NET.189.0	User network access

Some of those results can be correlated with Les Gordon GCIA assignment: http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc

Since some of the signatures were apparently logging traffic for a specific host and port, we also used the inbound traffic alerts destination ports when the port was requested multiple times (to get rid of the scans entries) or if the destination port was 137 (not necessarily controlled connection attempt).

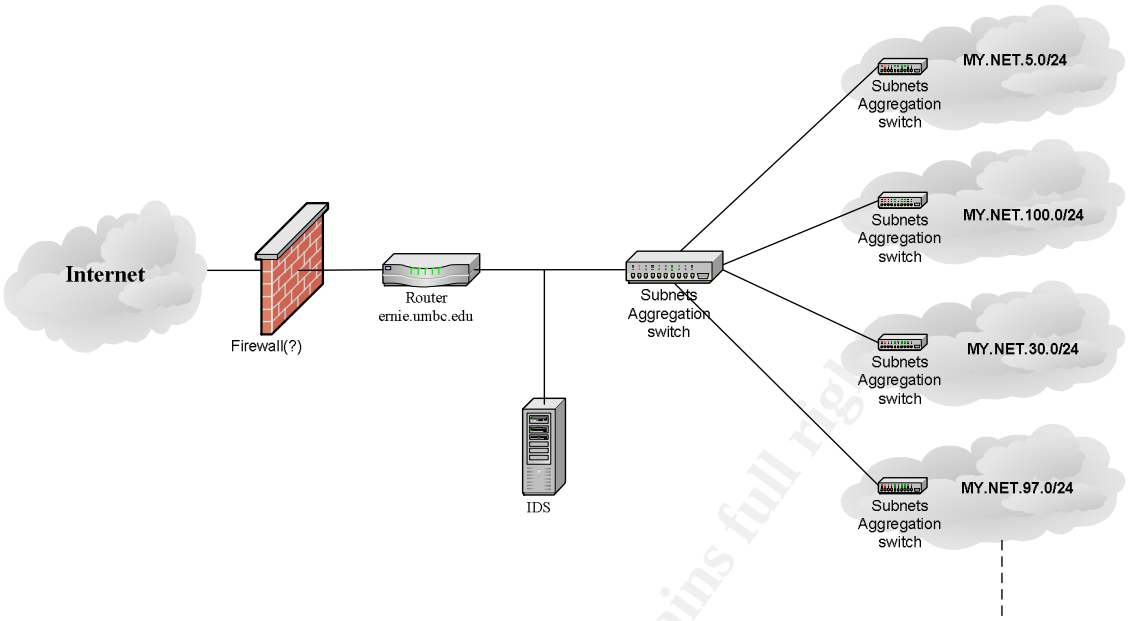
Some general observations:

- The table above lists only the servers that generated traffic from a port lower than 1024. It is very likely that that list above is not exhaustive.
- Over 5 days we saw 556 occurrences of the following alarm "TCP SRC and DST outside network" with IPs belonging to common ISPs (AOL, Comcast, …). This could be caused by:
    - The use of source routing
    - A bad dialup or VPN client configuration with a bad static route
    - Spoofing

  The fact that most of the source hosts for those messages are coming from AOL (dial-up) and Comcast on a segment registered in Baltimore, MD (local to the university) would confirm the hypothesis of a bad VPN or dialup configuration.

- We found OOS packets coming from the internal segment going to internal segment (MY.NET.70.234 to MY.NET.16.174) which lead us to think that the IDS is located on the different LANs exchange point and therefore behind the university firewall (or router if there is no firewall).
- A first analysis of the logs reveals that the university is monitoring any traffic with the following destination addresses:
    - MY.NET.30.3
    - MY.NET.30.4
    - MY.NET.100.165 for port HTTP and FTP
- Most of the /24 x.x.x.1 IP addresses resolves to the same hosts: ernie.umbc.edu indicating that it is very likely that a unique router is the gateway for those subnets.
- We did not see any ICMP/UDP traffic in neither of the scan, alert or OOS files which lead us to think that the different subnets are protected by a firewall and/or ACLs on the gateway router.

The following diagram outlines the network architecture:



Among all the internal IP listed in the logs, we isolated the following three hosts with custom, dedicated signature. Our guess is that the administrators implemented those rules to log any traffic targeting those systems either because it was an easy way to gather usage statistics (number of connections, sources of connections) or because those servers were sensitive to the university.

MY.NET.30.3: (514/tcp)
MY.NET.30.4: (524/tcp, 8009, 51443, 80; rest minor)
Both of those servers are running a Novel Netware server suite.
MY.NET.100.165
This server is the computer science and electrical engineering web and FTP server.

4. Detects

The following table lists the number of alarms that occurred during the 5 days. We will focus on the alarms with more than 5000 occurrence during the 5 days.

Alarm message	Occurrence
CS WEBSERVER - external web traffic	130536
High port 65535 tcp - possible Red Worm - traffic	67019
SMB Name Wildcard	53273
MY.NET.30.4 activity	37097
spp_http_decode: IIS Unicode attack detected	32170
Queso fingerprint	10568
MY.NET.30.3 activity	7698
spp_http_decode: CGI Null Byte attack detected	5051

EXPLOIT x86 NOOP	4061
SYN-FIN scan!	2555
Tiny Fragments - Possible Hostile Activity	1426
connect to 515 from outside	1260
IDS552/web-iis_IIS ISAPI Overflow ida nosize	920
SUNRPC highport access!	896
High port 65535 udp - possible Red Worm - traffic	795
NMAP TCP ping!	740
Null scan!	640
TCP SRC and DST outside network	543
TCP SMTP Source Port traffic	465
Possible trojan server activity	440
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	390
Incomplete Packet Fragments Discarded	382
[UMBC NIDS IRC Alert] IRC user /kill detected possible trojan.	294
SNMP public access	288
External RPC call	222
NIMDA - Attempt to execute cmd from campus host	157
SMB C access	156
FTP passwd attempt	136
EXPLOIT x86 stealth noop	74
FTP DoS ftpd globbing	72
TFTP - Internal TCP connection to external tftp server	69
TFTP - Internal UDP connection to external tftp server	68
EXPLOIT x86 setuid 0	65
EXPLOIT x86 setgid 0	48
CS WEBSERVER - external ftp traffic	46
IRC evil - running XDCC	44
EXPLOIT NTPDX buffer overflow	42
Notify Brian B. 3.54 tcp	34
Notify Brian B. 3.56 tcp	33
Attempted Sun RPC high port access	28
RFB - Possible WinVNC - 010708-1	21
TFTP - External TCP connection to internal tftp server	13
ICMP SRC and DST outside network	13
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	10
TFTP - External UDP connection to internal tftp server	10
NIMDA - Attempt to execute root from campus host	10
External FTP to HelpDesk MY.NET.70.50	10
Back Orifice	10
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	9
Probable NMAP fingerprint attempt	8
connect to 515 from inside	6
Traffic from port 53 to port 123	6
External FTP to HelpDesk MY.NET.70.49	4
NETBIOS NT NULL session	3
DDOS shaft client to handler	3
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	2

[UMBC NIDS IRC Alert] K\:line'd user detected	2
External FTP to HelpDesk MY.NET.53.29	2
EXPLOIT VQServer admin	2
DDOS mstream client to handler	2
[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot	1
DDOS mstream handler to client	1

Attack#1: CS WEBSERVER - external web traffic

Traffic flow - Incoming: 130536 Outgoing: 0
Hosts - Ext SRCs: 19995 Int DSTs: 1 Int SRCs: 0 Ext DSTs: 0
Standard Snort SIDs: None - custom alert

This rule has probably been setup to monitor and count the number of hosts trying to reach the web server. The signature actually alarms on legitimate traffic.

```
alert tcp $EXTERNAL_NET any -> MY.NET.100.165 80 (msg:"CS WEBSERVER –
external web traffic"; classtype:misc-activity;)
```

Correlation: The alarm was seen in October 2002 by Edward Peck in his GCIA assignment:
http://www.giac.org/practical/Edward_Peck_GCIA.doc
Michael Dawson reported the same message was seen in December 2001:
http://www.giac.org/practical/Wade_Walker_GCIA.doc

Recommendation: this rule is fairly noisy; we would not recommend using the IDS to count the number of packets of connections to a server. The short logs would not allow an efficient root-cause analysis should the web server be compromised on port 80. The rule can also be potentially dangerous, and inexperienced administrator might, by reordering the signatures, shadow a more specific rule and make the IDS blind to lethal attacks.

Attack#2: High port 65535 tcp - possible Red Worm - traffic

Traffic flow - Incoming: 34251 Outgoing: 32768
Hosts - Ext SRCs: 1015 Int DSTs: 51 Int SRCs: 40 Ext DSTs: 80
Standard Snort SIDs: None - custom alert

This alert is also a custom alert, snort alerts on packet coming or going to port 65535. The snort alert would look like this:

```
alert tcp $EXTERNAL_NET any -> $INTERNAL_NET any (msg:" High port 65535
tcp - possible Red Worm - traffic"; classtype:misc-activity;)
```

Port 65535/tcp is used by a well know Trojan: Adore also named Red Worm is a linux trojan with the ability to propagate automatically by scanning randomly remote hosts for known vulnerability. Port 65535 is also a valid source port for a client. However we noticed that compared to older GCIA assignments the number of "high port" attacks increased tremendously. Les Gordon in December 2002 counted only 13 incoming and 13 outgoing flows.

Recommendation: Just like for Attack#1, we would not recommend to create a generic signature based solely on a single TCP parameter. This configuration generates lots of false positive and may hide more dangerous activities.

Attack#3: SMB Name Wildcard

Traffic flow - Incoming: 53,273 Outgoing: 0
Hosts - Ext SRCs: 19995 Int DSTs: 1290 Int SRCs: 0 Ext DSTs: 0
Standard Snort SIDs: No match

This alarm signal udp traffic trying on destination port 137. The snort rule would look like this:

```
alert udp any any -> $INTERNAL_NET 137 (msg:"SMB Name Wildcard"; content:"CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|0000|"; classtype:misc-activity)
```

Once more, this signature is very generic. Windows explorer can by default send a packet on any available interface with any of the IP address configured in the server to access to hosts. For example, when a remote user deactivates its VPN after connecting to the university internal network would keep sending SMB requests to the university hosts. The VPN tunnel being down, the requests would then show up on the front-end router instead of the VPN server. This traffic is seen very frequently on firewalls and is usually rarely something to be concerned about.

Correlation: Les Gordon GCIA assignement:
http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc

Recommendation: even if the alert can sometimes be useful for forensic analysis, the number of alarms generated may affect the IDS performance.

Attack#4: MY.NET.30.4 activity

Traffic flow - Incoming: 37,097 Outgoing: 0
Hosts - Ext SRCs: 431 Int DSTs: 1 Int SRCs: 0 Ext DSTs: 0
Standard Snort SIDs: None - custom alert

This alert can be compared with the alert #1. The alarm is generated if a packet has MY.NET.30.4 for destination IP and is not coming from the internal network.

```
alert any !$INTERNAL_NET any -> 130.85.30.4 any (msg:"MY.NET.30.4
activity"; classtype:misc-activity)
```

Unlike alarm#1, the signature seems to target any ports for that specific IP. As we mention in the first section this server is running novel NetWare 6.0. The server is accessed through the following ports:

Occurrence	Port	Comments
15408	8009	Front page of the Net storage server
14620	51443	Net storage: web access to a document repository
4323	524	Common port used by Novell to provide a shell access to a server. The intent here is once more to log all the traffic to a particular server
2715	80	Front page of the Net storage server

(grep "MY.NET.30.4 activity" alert.total.csv | cut -d, -f9 | sort | uniq)

A user typically access to port 80 or 8009 and is redirected to port 51443 for web authentication and access to the files.

The host has been accessed by 431 distinct hosts. An analysis of the source IP addresses reveals that only the following hosts accessed the document repository:

152.16.118.216
172.155.65.52
24.35.42.249
68.48.217.68
68.48.57.29
68.54.93.211
68.55.232.108
68.55.27.218
68.55.71.120

Only 68.48.217.68 came through the front page served from port 8009, and 406 different source IP addresses accessed the server on port 80. Among them 398 did not follow to port 51443. Inktomi web crawler is actively scanning the server from various IP addresses. As quick search in altavista with the following parameters: "Welcome to NetWare 6" url:umbc showed that the Novell server is actually referenced in the search engine.

Recommendation: I would definitely remove the port 80 access on the server to prevent web crawlers to reference the site. Vulnerabilities have been discovered in Apache/1.3.27 (used by Novell Netware 6.0) explained in the following CVE vulnerability:

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0460

as well as some other bugs mentioned in the apache 1.3 chanlog file:

http://www.apache.org/dist/httpd/CHANGES_1.3

A simple search in a search engine would therefore list MY.NET.30.4 as been vulnerable to an attack on Novell netware 6.0.

Attack#5: spp_http_decode: IIS Unicode attack detected

Traffic flow - Incoming: 2101 Outgoing: 30069
Hosts - Ext SRCs: 208 Int DSTs: 231 Int SRCs: 372 Ext DSTs: 684
Standard Snort SIDs: None – http_decode preprocessor alert

The IIS Unicode attack is well known since it was used as the propagation mean by code red. The attack consists in using UNICODE encoded characters to execute commands or read restricted access documents on an IIS4.0 and ISS5.0 server. The attack in referenced in CVE:

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884
In bugtraq:
http://www.securityfocus.com/archive/1/140091/2003-04-13/2003-04-19/2

Snort preprocessor alert if a URL contains one more of those Unicode encoded characters '.','/','\'.

The signature in the logs analyzed have been modified to alarm on "any" destination IP addresses.

The signature is known to generate false positive for encrypted website. Asian character sets or automatically generated tracking cookies can also cause the IDS to alarm. The IP addresses of the top 5 talkers for this alert included a chat portal and several sites located in China and Korea.

Correlation: This attack has been seen in many GCIA assignments. We noted from the previous analyses that the amount of alarms on inbound traffic decreased significantly from most of the previous assignments. Les Gordon in august 2002 reported 48234 alarms (http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc) and Al Maslowski reported 42440 alerts in December 2002 (http://www.giac.org/practical/GCIA/Al_Maslowski-Yerges_GCIA.pdf).

Attack#6: Queso fingerprint

Traffic flow - Incoming: 10568 Outgoing: 0
Hosts - Ext SRCs: 323 Int DSTs: 82 Int SRCs: 0 Ext DSTs: 0
Standard Snort SIDs: None - custom alert

Queso is a utility sending crafted IP packets to remote system in order to determine various characteristics of the remote operating system. The following links give more information about queso:

http://www.iss.net/security_center/advice/Intrusions/2000321/default.htm
or
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0454

Fingerprinting is used at the early stage of hacking attempt to gather version and system properties prior to an attack. Therefore, fingerprint should be taken seriously as a proactive security measure.

The IDS is really reporting here that odd packets have been sent on the network that might be used for fingerprinting purpose. *Queso* is just one of the numerous fingerprinting tools openly available. Queso has the property to send SYN packets with the two reserved bits set. The snort signature would look like this;

alert tcp any any -> any any (msg:"Queso fingerprint";flags: S12;)

As the above statistics indicates, no *Queso* fingerprinting activity was initiated internally. The network has been scanned 10568 times by 323 different hosts. To determine if the scans were coordinated we sorted the source addresses by /24 (Class C) subnets:

Occurence	Subnet source
4142	216.95.201
1045	193.41.64
985	141.152.40
964	209.47.197
891	217.9.225
339	66.48.78
248	204.92.158
182	213.186.35
102	12.255.198
91	207.228.236

```
*command used: grep -i "queso" alert.total.csv | cut -d, -f6 | grep -v MY.NET | cut -d. -
                                                f1,2,3 | sort | uniq -c | sort -rn | head -10
```

41% of the fingerprinting activity is coordinated between 18 hosts in the same subnet source: 216.95.201. Since the subnet is registered to UUnet Canada (see whois query bellow) I would recommend to open a case with UUnet to have this investigated. I would also recommend opening a case with the other subnet owner above.

Finally we would also recommend blocking (if the upstream networking equipment allows it) and packet with both reserved bit set to prevent any queso fingerprint activity.

Attack#7: MY.NET.30.3 activity

Traffic flow - Incoming: 7698 Outgoing: 0
Hosts - Ext SRCs: 67 Int DSTs: 5 Int SRCs: 0 Ext DSTs: 0
Standard Snort SIDs: None - custom alert

Unlike My.NET.30.4 (attack #4) My.NET.30.3 do not have the same front end interface and therefore is not susceptible to be listed in a search engine. However, the rule setup is once too generic and generates lots of false positive.

Attack#8: spp_http_decode: CGI Null Byte attack detected

Traffic flow - Incoming: 166 Outgoing: 4885
Hosts - Ext SRCs: 6 Int DSTs: 1 Int SRCs: 95 Ext DSTs: 119
Standard Snort SIDs: None - custom alert

This alert is generated by the snort http decode pre-processor if a URL contains a %00 character. The attack is referenced as CVE-2000-0149:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0149

The vulnerability allows a remote attacker to read critical system file using a CGI script named 'web_store.cgi' bug. Roy Naldo analysed this attack in his GIAC assignment:
http://www.giac.org/practical/Roy_Naldo_GCIA.zip

This alarm can generate many false positive, some web site using user tracking cookies generated randomly can uses %00 is their URL. Snort FAQ mentioned "Sometimes you may see false positives with sites that use cookies with urlencoded binary data, or if you're scanning port 443 and picking up SSLencrypted traffic".

The alerts seen here are very likely to be false positive. Note that by itself, the alarm does not allow to conclude for sure. However, the IP addresses we are seeing for that attack can be correlated with other attacks to see if they are part of a broader activity.

5. Top talker list

As we mentioned earlier, the university is logging the traffic for some specific host and port in the alert file. Those alarms do not carry lots of information in the context of an audit and will therefore be filtered out. (MY.NET.30.3 activity, MY.NET.30.4 activity and CS WEBSERVER - external web traffic)

We will focus on the rest of the alarms present in the alarm file. We sorted the alarms by order of occurrences.

```
$ grep -vi "CS WEBSERVER - external web traffic" alert.total.csv | grep -v "MY.
NET.30.4 activity" | grep -v "MY.NET.30.3 activity" | cut -d, -f6 | grep -v "MY
.NET" | cut -d, -f6 | sort | uniq -c | sort -rn | head -10
```

Rank #	Number of occurence	IP address	Alerts
1	33552	81.48.143.73	High port 65535 tcp - possib Worm - traffic
2	33323	MY.NET.84.216	High port 65535 tcp - possib Worm - traffic
3	5910	169.254.45.176	SMB Name Wildcard

4	2551	66.82.245.45	SYN-FIN scan!
5			spp_http_decode: IIS U
	2270	MY.NET.153.153	attack detected
6	1977	64.228.212.245	SMB Name Wildcard
7	1624	64.228.213.12	SMB Name Wildcard
8	1513	64.228.214.41	SMB Name Wildcard
9			spp_http_decode: IIS U
			attack detected
	1504	MY.NET.97.183	IRC user /kill detected
10			spp_http_decode: IIS U
	1362	MY.NET.97.16	attack detected

6. five external source registration info

Address #1: 216.95.201.15

This host was the first talker in the attack #6 “queso fingerprinting”.

```
OrgName:    UUNET Technologies, Inc.
OrgID:      UU
Address:    22001 Loudoun County Parkway
City:       Ashburn
StateProv:  VA
PostalCode: 20147
Country:    US

NetRange:   216.94.0.0 - 216.95.255.255
CIDR:       216.94.0.0/15
NetName:    UUNETCA6-A
NetHandle:  NET-216-94-0-0-1
Parent:     NET-216-0-0-0-0
NetType:    Direct Allocation
NameServer: NS.UUNET.CA
NameServer: NS2.UUNET.CA
NameServer: AUTH01.NS.UU.NET
Comment:
RegDate:
Updated:    2002-05-21

TechHandle: UC24-ORG-ARIN
TechName:   UUNET Canada Registrar
TechPhone:  +1-888-886-3865
TechEmail:  registrar@uunet.ca

OrgAbuseHandle: ABUSE3-ARIN
OrgAbuseName:   abuse
OrgAbusePhone:  +1-800-900-0241
OrgAbuseEmail:  abuse-mail@mci.com

OrgNOCHandle: OA12-ARIN
OrgNOCName:   UUnet Technologies, Inc., Technologies
OrgNOCPhone:  +1-800-900-0241
OrgNOCEmail:  help4u@mci.com

OrgTechHandle: SWIPP-ARIN
OrgTechName:   swipper
OrgTechPhone:  +1-800-900-0241
OrgTechEmail:  swipper@uu.net

# ARIN WHOIS database, last updated 2003-08-22 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database
```

Address #2: 66.82.245.45

This IP has performed during those 5 days a syn-fin scan on port 21/TCP of more than 2554 internal IP addresses. We actually found that that IP was already reported in DSHIELD for the same 21/tcp scan:

http://www.dshield.org/ipinfo.php?PHPSESSID=85ad5f154ea89d1804f2c1b12227cf99&ip=66.82.245.45&Submit=Submit

```
OrgName:    Hughes Network Systems
OrgID:      HNS
Address:    11717 Exploration Lane
Address:    DirecWAY Network Management Center
Address:    attn: Network Security Manager
City:       Germantown
StateProv:  MD
PostalCode: 20876
Country:    US
NetRange:   66.82.0.0 - 66.82.255.255
CIDR:       66.82.0.0/16
NetName:    DIRECPC-1BLK
NetHandle:  NET-66-82-0-0-1
Parent:     NET-66-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.DIRECPC.COM
NameServer: NS2.DIRECPC.COM
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    2001-02-28
Updated:    2003-01-21
TechHandle: ZD63-ARIN
TechName:   DirecPC
TechPhone:  +1-301-601-7205
TechEmail:  abuse@direcpc.com
OrgTechHandle: NSM5-ARIN
OrgTechName:   Network Security Manager
OrgTechPhone:  +1-301-601-7205
OrgTechEmail:  abuse@direcpc.com
```

Address #3: 81.48.143.73

First talker for our “High port 65535 tcp - possible Red Worm – traffic” (top talker list)

```
inetnum:      81.48.143.0 - 81.48.143.255
netname:      IP2000-ADSL-BAS
descr:        BSPUT108 Puteaux Bloc2
country:      FR
admin-c:      WITR1-RIPE
tech-c:       WITR1-RIPE
status:       ASSIGNED PA
remarks:      for hacking, spamming or security problems send mail to
remarks:      postmaster@wanadoo.fr AND abuse@wanadoo.fr
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20020710
changed:      gestionip.ft@francetelecom.com 20030318
source:       RIPE

route:        81.48.0.0/16
descr:        France Telecom
descr:        Wanadoo Interactive
origin:       AS3215
remarks:      ------------------------------------------
remarks:      For Hacking, Spamming or Security problems
remarks:      send mail to      abuse@francetelecom.net
remarks:      ------------------------------------------
notify:       addr-reg@rain.fr
mnt-by:       RAIN-TRANSPAC
changed:      tfischer@rain.fr 20020702
source:       RIPE

role:         Wanadoo Interactive Technical Role
address:      WANADOO INTERACTIVE
address:      48 rue Camille Desmoulins
address:      92791 ISSY LES MOULINEAUX CEDEX 9
address:      FR
phone:        +33 1 58 88 50 00
e-mail:       abuse@wanadoo.fr
e-mail:       technical.contact@wanadoo.com
admin-c:      WITR1-RIPE
tech-c:       WITR1-RIPE
nic-hdl:      WITR1-RIPE
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20010504
changed:      gestionip.ft@francetelecom.com 20010912
changed:      gestionip.ft@francetelecom.com 20011204
changed:      gestionip.ft@francetelecom.com 20030428
       source:       RIPE
```

Address #4: 218.145.25.112

This IP address is the only source IP address that alarmed on the “Notify Briaan B.” alert.

```
inetnum:      218.144.0.0 - 218.159.255.255
netname:      KORNET
descr:        KOREA TELECOM
descr:        Network Management Center
country:      KR
admin-c:      DL248-AP
tech-c:       GK40-AP
remarks:      ****************************************
remarks:      Allocated to KRNIC Member.
remarks:      If you would like to find assignment
remarks:      information in detail please refer to
remarks:      the KRNIC Whois Database at:
remarks:      http://whois.nic.or.kr/english/index.html
remarks:      ****************************************
mnt-by:       MNT-KRNIC-AP
mnt-lower:    MNT-KRNIC-AP
changed:      hostmaster@apnic.net 20010924
status:       ALLOCATED PORTABLE
source:       APNIC
```

The IP has been reported for attack port 80 and is listed as an anonymous proxy server:

http://theone.ru/proxy/proxys15.html

This URL describes the attack:

http://archives.neohapsis.com/archives/incidents/2003-08/0097.html

Try to find the match in our logs:

```
scans.030728:Jul  28  05:29:53  218.145.25.107:53028  ->  130.85.100.165:80  SYN
******S*
```

Address #5: 193.252.203.96

```
inetnum:      193.252.203.0 - 193.252.203.255
netname:      IP2000-ADSL-BAS
descr:        France Telecom IP2000 ADSL BAS
descr:        BSNAN102 Nantes Bloc1
country:      FR
admin-c:      FDTR1-RIPE
tech-c:       FDTR1-RIPE
status:       ASSIGNED PA
remarks:      for hacking, spamming or security problems send mail to
remarks:      postmaster@wanadoo.fr AND abuse@wanadoo.fr
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20010213
changed:      gestionip.ft@francetelecom.com 20010517
changed:      gestionip.ft@francetelecom.com 20030318
source:       RIPE

route:        193.252.128.0/17
descr:        France Telecom
origin:       AS3215
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20010613
source:       RIPE

role:         FTLD-IAN Domain Technical Role
address:      France Telecom Long Distance
address:      IP & ATM Network
address:      3 avenue Francois Chateau
address:      35000 RENNES
address:      FR
e-mail:       noc@francetelecom.net
admin-c:      HC253-RIPE
tech-c:       HC253-RIPE
nic-hdl:      FDTR1-RIPE
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20010213
changed:      gestionip.ft@francetelecom.com 20030219
source:       RIPE
```

7. Internal activity: P2P and files transfer

Destination Port	Service	Occurrences	Internal hosts involved	External hosts involved
4665	EDonkey 2000	0	0	0
6346	GNUTella	5	1	0
4662	EDonkey 2000	30	8	8
1214	KaZaA	23	9	19 (4 ext SRC IP)
6665-6666-6667	IRC	52	8	4
5500-5001	Hotline	0	0	0
8311	Scour	0	0	0
8888	AudioGalaxy	18	5	1
6257 – 6699	WinMX	255	4	31

Source: http://honor.trusecure.com/pipermail/firewall-wizards/2001-September/011235.html

Those numbers represent bandwidth used by file sharing activity using P2P client but does not include HTTP and FTP non-work related downloads. In addition to the high bandwidth consumption, a significant amount of programs accessible from various P2P hosts are infected by viruses and Trojans threatening the university network.

We will focus here on internal equipment activities, particularly from a file sharing prospective. The administrator is apparently aware of the issue because he configured a set of signatures to isolate file sharing activities. The table bellow lists all the IRC related alerts with the associated number of occurrence:

nb	Alert message
294	294 [UMBC NIDS IRC Alert] IRC user /kill detected , possible trojan
44	44 IRC evil - running XDCC
10	10 [UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.
9	9 [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC
2	2 [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC
2	2 [UMBC NIDS IRC Alert] K\:line'd user detected
1	1 [UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC

According to the alert description, the alarm is generated if an IRC /kill command is received by an internal host. This alert, if indicating that and internal user is using IRC, does not necessarily indicate that a Trojan is installed.

The second most frequent alert on the other hand, is looking for the keyword “XDCC” in the packet. XDCC is commonly used for file sharing over IRC.

```
grep "running XDCC" alert.total.csv | cut -d, -f6,8 | sort | uniq

listed two hosts:
MY.NET.198.221,205.188.149.12 (undernet.irc.aol.com)
MY.NET.74.216,212.161.35.251
```

MY.NET.198.221 alerts analysis reveals that it received 5 XDCC send request from the 205.188.149.12 server which confirms that this system is running an XDCC bot or has been compromised. We also noted that the host requested for a flie offered via XDCC indicating that a user is actually connected to this server.

```
$ grep "Possible Incoming XDCC Send Request Detected" ../alert.030* | grep MY.N
ET.198.221 | more
../alert.030728:07/28-02:35:23.323290  [**] [UMBC NIDS IRC Alert] Possible Incom
ing XDCC Send Request Detected. [**] 205.188.149.12:6667 -> MY.NET.198.221:1026
../alert.030728:07/28-06:52:11.479179  [**] [UMBC NIDS IRC Alert] Possible Incom
ing XDCC Send Request Detected. [**] 205.188.149.12:6667 -> MY.NET.198.221:1026
../alert.030728:07/28-12:26:26.649615  [**] [UMBC NIDS IRC Alert] Possible Incom
ing XDCC Send Request Detected. [**] 205.188.149.12:6667 -> MY.NET.198.221:1026
../alert.030728:07/28-16:10:50.175683  [**] [UMBC NIDS IRC Alert] Possible Incom
ing XDCC Send Request Detected. [**] 205.188.149.12:6667 -> MY.NET.198.221:1026
../alert.030728:07/28-18:49:21.491465  [**] [UMBC NIDS IRC Alert] Possible Incom
ing XDCC Send Request Detected. [**] 205.188.149.12:6667 -> MY.NET.198.221:1026
```

Beyond the XDCC activities, 193.252.203.96 attempted to access to MY.NET.198.221 65535 TCP port as indicated bellow:

```
$ grep MY.NET.198.221 alert.total.csv | cut -d, -f5,6,8 | sort | uniq -c | sort
 -rn
     13 IRC evil - running XDCC,MY.NET.198.221,205.188.149.12
      5     [UMBC     NIDS     IRC     Alert]     Possible     Incoming     XDCC     Send     Request
Detected.,205.188.149.12,MY.NET.198.221
      3 TFTP - External TCP connection to internal tftp server,MY.NET.198.221,193.252.203.96
      2 TFTP - External TCP connection to internal tftp server,193.252.203.96,MY.NET.198.221
      2 High port 65535 tcp - possible Red Worm - traffic,MY.NET.198.221,193.252.203.96
      2 High port 65535 tcp - possible Red Worm - traffic,193.252.203.96,MY.NET.198.221
      2 DDOS mstream client to handler,193.252.203.96,MY.NET.198.221
      1 [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC,MY.NET.198.221,205.188.149.12
        1 SUNRPC highport access!,193.252.203.96,MY.NET.198.221
```

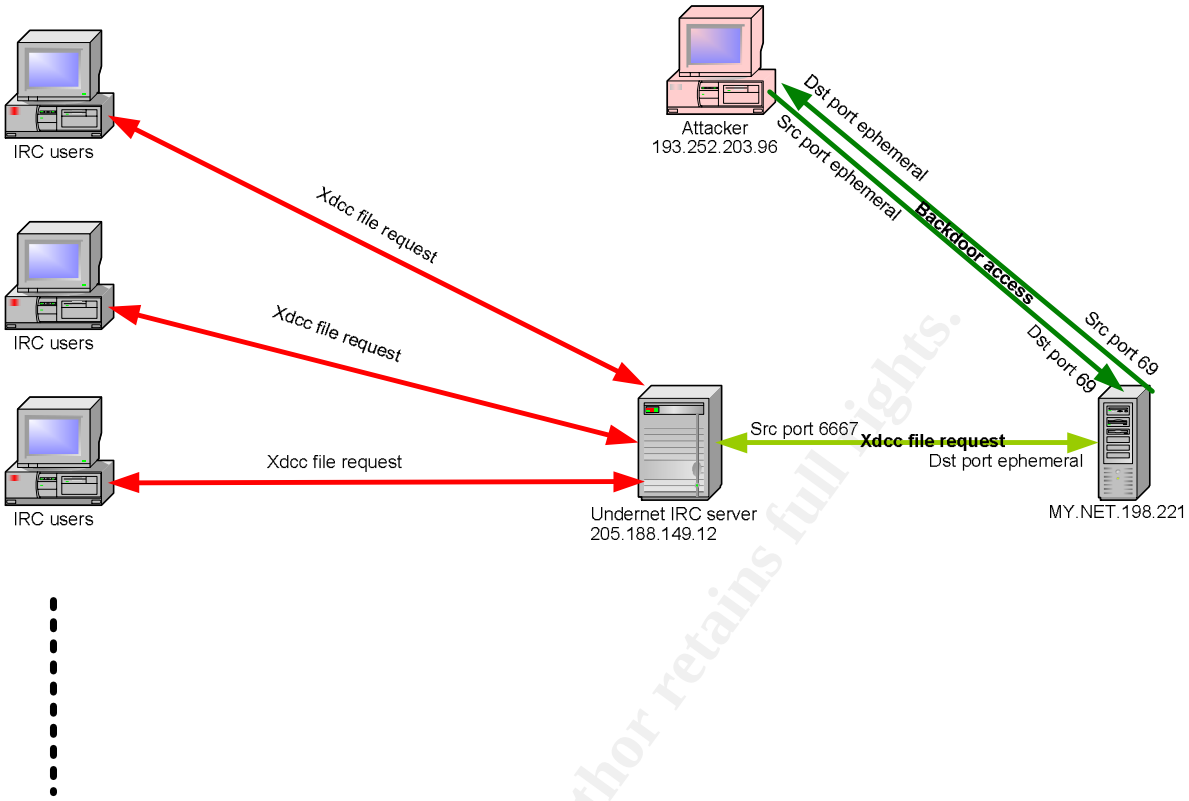
193.252.203.96 tried to access to MY.NET.198.221 on various port. MY.NET.198.221 replied on port 69/tcp (port usually used by tftp with UDP traffic only) and 65535/tcp.

```
$ grep 193.252.203.96 ../alert.030* | grep -v portscan
../alert.030730:07/30-01:18:13.795168  [**] TFTP - External TCP connection to internal
tftp server [**] 193.252.203.96:2087 -> MY.NET.198.221:69
../alert.030730:07/30-01:18:13.795390  [**] TFTP - External TCP connection to internal
tftp server [**] MY.NET.198.221:69 -> 193.252.203.96:2087
../alert.030730:07/30-01:19:50.706996  [**] TFTP - External TCP connection to internal
tftp server [**] 193.252.203.96:3273 -> MY.NET.198.221:69
../alert.030730:07/30-01:19:50.707357  [**] TFTP - External TCP connection to internal
tftp server [**] MY.NET.198.221:69 -> 193.252.203.96:3273
../alert.030730:07/30-01:19:51.729293  [**] TFTP - External TCP connection to internal
tftp server [**] MY.NET.198.221:69 -> 193.252.203.96:3273
../alert.030730:07/30-02:04:58.813276  [**] SUNRPC highport access! [**]
193.252.203.96:3854 -> MY.NET.198.221:32771
../alert.030730:07/30-01:41:05.087700  [**] DDOS mstream client to handler [**]
193.252.203.96:2676 -> MY.NET.198.221:15104
../alert.030730:07/30-01:41:06.272636  [**] DDOS mstream client to handler [**]
193.252.203.96:2676 -> MY.NET.198.221:15104
../alert.030730:07/30-02:47:56.948508  [**] High port 65535 tcp - possible Red Worm -
traffic [**] 193.252.203.96:2896 -> MY.NET.198.221:65535
../alert.030730:07/30-02:47:56.948799  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.198.221:65535 -> 193.252.203.96:2896
../alert.030730:07/30-02:47:57.543485  [**] High port 65535 tcp - possible Red Worm -
traffic [**] 193.252.203.96:2896 -> MY.NET.198.221:65535
../alert.030730:07/30-02:47:57.543993  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.198.221:65535 -> 193.252.203.96:2896
```

In the packets above, we can see that an external system accessed the server on port 65535 and 69 and got a reply from MY.NET.198.221.Because the first packet seen is actually coming from an external IP the packets seen above are even more suspicious.

We have not find any traces of scan or OOS packet from that IP or to the internal system in the scans or OOS packets.

The second internal IP involved in the traffic: MY.NET.74.216 received also 5 XDCC requests, we have not seen any XDCC file being requested from the internal server.



Suspicious IRC/TFP activity to MY.NET.198.221 link graph

8. Insight on compromised/ dangerous activity

8.1 The Notify Brian signature

We would recommend to remove the “Notify Brian B” rules. Because of rule ordering this rule might shadow another, more important rule in the system depending on hoe the rules are organized in the system.

E.g.:

07/29-01:36:49.064710 [**] Notify Brian B. 3.54 tcp [**] 172.138.51.190:3007 -> 255.255.3.54:17300

1730 is actually a Trojan port: 17300/TCP kuang2

same thing with “activity MY.NET.X.X”. Should actually be filtered = generate too much alarms and hide the real problem.

9. description of my analysis process

We first converted the scan file into a csv formatted file using the excellent alert to csv perl converter from Les Gordon:

http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc

All our analysis has been based on the csv file and unix commands that you have seen in the analysis.