Global Information Assurance Certification Paper

SANS Intrusion Detection in Depth

GCIA Practical Assignment

Version 3.3

Antony Gummery

Submitted by: Antony Gummery Date Submitted: 05/04/2003

Table of Contents

Abstract

This assignment v3.3 is a requirement of the SANS GIAC GCIA certification track. It is broken down into three main parts:

1. **Describe the state of Intrusion detection**

   This section is a white paper on the subject of IDS Event Data Management.

   This paper will discuss, the challenges faced in dealing with the event data outputted from Intrusion Detection Systems, and review some of the tools and techniques being used to overcome these challenges.

2. **Network Detects**

   This section involves analysing three network detects and fully documenting the analysis process and any conclusions drawn.

   Two detects were submitted to the intrusions@incidents.org mailing list and cross examined by the SANS community. Comments and replies to these submissions are included in this section.

3. **Analyze This**

   This is a scenario based activity.

   A security audit must be carried out on a set of log data supplied by a University including full analysis reporting, and conclusions to be drawn on the state of the campus network systems.

Part 1 - Describe the State of Intrusion Detection

IDS Event Data Management

Summary

Ever since Intrusion detection systems, (IDS), were first deployed, whether for an individual responsible for a single sensor, or a Managed Security Services Provider (MSSP) responsible for many different types of sensors, one of the biggest problems has always been how to handle the sometimes vast amounts of event data generated by these devices.

Even today, dealing with all of this data is still perhaps cited by most sources, as the number one drawback with deploying IDS. In the commercial market, more and more emphasis is now being placed on the handling of event data, both by the IDS vendors themselves and other organisations developing software and services for IDS.

This paper will discuss, the challenges faced in dealing with the event data outputted from Intrusion Detection Systems, and review some of the tools and techniques being used to overcome these challenges.

For the purposes of this paper, the subject of handling event data will be termed Event Data Management.

IDS Development

The first commercially available IDS began to appear just after the mid 1990's, from companies like Internet Security Systems (ISS), with the release of their RealSecure network sensor. Early emphasis in the development of commercial IDS technology was placed on recognising known malicious activity. This was achieved using a technique called misuse detection, and involved comparing network data packets against a set of pre-defined signatures or rules, developed for known attack behaviour and tools. The importance of the word 'known' in the previous sentence cannot be underestimated. Signatures or rules can only be developed against background knowledge of known vulnerabilities and exploits. Although there is scope to anticipate to some extent, what attempts could be made to attack devices, this doesn't translate into pre-emptive Intrusion Detection Systems. In reality, signature based IDS are always playing catch up to newly discovered vulnerabilities and exploits, and are thus, more a reactionary security tool. In this respect they are often compared to anti-virus software.

Despite some of the shortcomings of this type of IDS, they still dominate today's commercial market place. Some of the most familiar products include, ISS RealSecure; Cisco Secure IDS; Enterasys Dragon; and Snort.

The other main type of IDS technology is referred to as anomaly detection. This system operates by analysing network traffic for deviations from the normal pattern of activity. It avoids the pattern-matching technique employed in misuse detection. Examples of where anomaly detection would be of benefit could include, alerting to someone logging into a system at an unusual time, or detecting a completely new worm exploit hitting your network. Although, it could be argued that these types of attacks could be detected by signature-based systems, this would require a human to perform extra analysis on the signature triggered events, in order to come to a similar conclusion as the anomaly based system would in a relatively short space of time.

For example, the logon scenario described above, could be detected by a signature based 'host' IDS, alerting to a user logon for a particular system. In order for this to be determined as suspicious, an analyst would need to correlate the event with a known logon policy for that particular system. A time consuming an inefficient process, much better suited to anomaly detection systems.

The second scenario discussed above regarding the detection a new worm exploit, would probably be detectable with a signature based system, despite this being a new attack, with no specific signature for detection of the attack installed on the IDS sensor. This would, perhaps be initially seen by an analyst, as port-scanning activity against a specific port, which could be further analysed to determine the nature of the attack being employed. Again this is time consuming and inefficient when compared to detection of the same attack by an anomaly based system.

There are drawbacks with both types of IDS detection technologies, in that either system still suffers from 'false-positives'. These are events triggered by an IDS, which are due to normal and acceptable behaviour, being misinterpreted. Signature-based systems are more prone to these, but that does not necessarily mean that they are ineffective. An ideal solution would be to have a hybrid system, which incorporated the benefits of both detection technologies, working together, via some form of correlation, to reduce false-positives and increase the overall protection offered by the system. Indeed the 'holy grail' of IDS technology would be to produce a system which detected both existing and new threats, for all network architectures, with a zero false-positive rate.

Coping with the data

Whilst the next generation IDS may well address many of the existing issues, for the time being we are still faced with the present main concern, that of dealing with the amount of event data generated from the current crop of commercially available systems. Whether a small IDS deployment using only 'out of the box' tools, or a MSSP, utilising proprietary or COTS delivery and analysis methods, this comes down to effective event data management.

This is a process which begins when the IDS sensor is first deployed on the network. Effective management of IDS event data should be a perpetual process which includes several main stages of data handling with continuous feedback into these stages, to improve the overall effectiveness of the system. One mechanism for achieving this is the Event Data Management Model (EDMM) shown in figure 1.

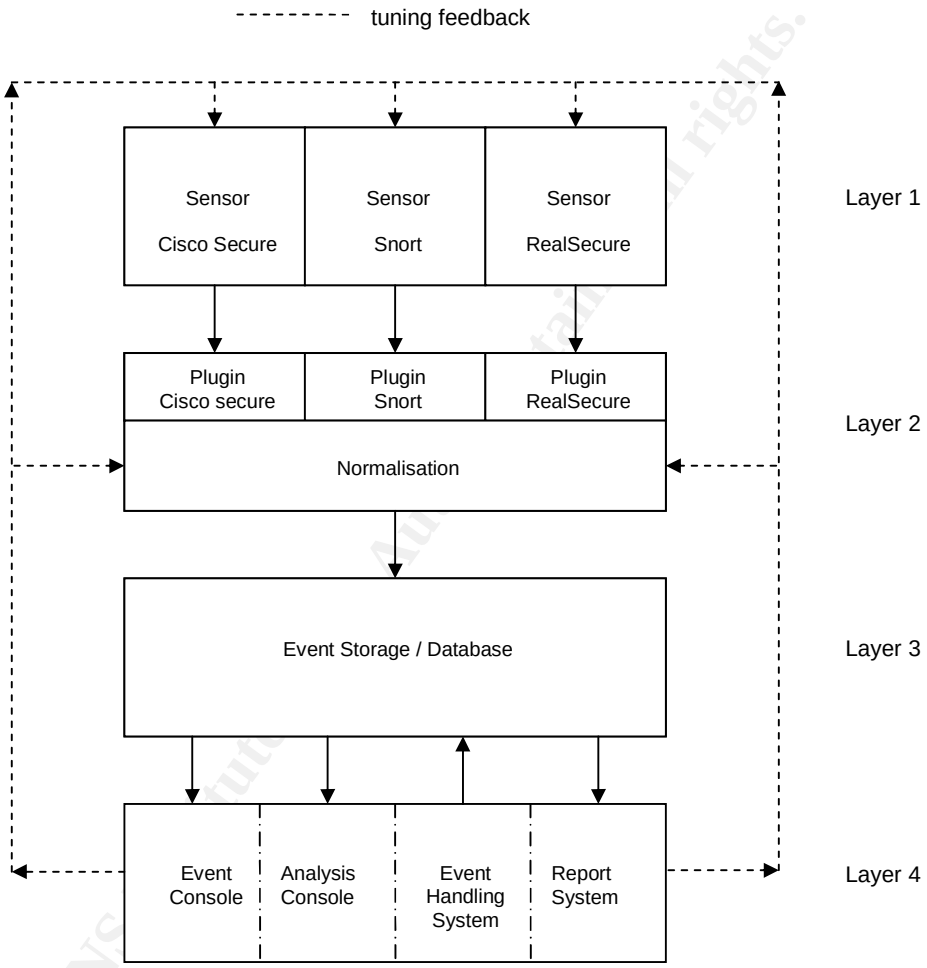


Figure 1

Event Data Management Model

Layer 1 - IDS Sensors

The IDS sensor would become a layer 1 device, and as many different vendor sensors can be deployed as there are available plugins for, in layer 2. The delivery mechanism from layer 1 to layer 2 can be anything appropriate, such as syslog, SNMP traps, proprietary methods, and utilising vpn's or dedicated kilostream links etc.

For the purposes of this paper we are focussing on IDS event data management, but it is also possible for any security device that is capable of outputting event or log data, to be deployed at layer 1. The only dependency on this would be the requirement for a device specific plugin to be available at layer 2.

Layer 2 – Event Normalisation, Correlation and Consolidation

This layer consists of two main stages. Firstly, in order for different sensor types to be used within the same overall mechanism, a plugin is required for each individual type of sensor. Acting as a translation process, the original format is presented in a way that is standard for the normalisation stage. This second stage of the event consolidation layer, allows scope for customising the format of event data as it is stored in the layer 3 database. The ability to manipulate the proprietary format of the event data outputted from the sensors, allows, for example, MSSP's and others to assign custom severity levels to events, and drop events altogether before they enter the database. This functionality can be further enhanced by the ability to provide this type of filtering against many criteria, such as source IP, destination IP, ports, event messages etc.

Layer 3 – Event storage / Database

Event data is the lifeblood of a functional IDS. Effective storage of this data is important for data integrity, forensic analysis and reporting, in the main. The schema of the database should be built around the normalisation stage of layer 2, and needs to be both robust, responsive to querying, and scalable.

Layer 4 – Event display, analysis and handling

This layer provides the front-end tools for event analysis, data forensics and if required, event handling. Every IDS deployment, whether big or small, requires some form of event handling, however this is more of a required feature built into the front-end analysis tools, used by larger deployments, such as those operated by a MSSP.

The overall mechanism of event data management, is not one that necessarily introduces any new concepts, but one that brings them all together into a simple layered structure, into which all facets of IDS deployment can be accommodated. The structure is flexible enough to allow a small deployment of a single type of IDS, using the proprietary tools and techniques in a single location, or alternatively, a larger, deployment of several different geographically dispersed IDS sensor types, using customised tools at layers 2, 3 or 4.

The omission of a management layer for the model is deliberate given that we are concentrating specifically on event data. However it is worth noting that to complete the picture, a management function must be accommodated within the overall deployment strategy. This stage could realistically fit anywhere within the model, as required, and as it is not directly involved in the mechanism of event data

management, it will not be discussed in any great detail, within the remainder of this paper.

As an example of how a proprietary IDS deployment, of a single sensor type fits in to the overall event data management model, the following section will describe the architecture of a commercially available system.

Example - ISS RealSecure

Internet Security Systems (ISS) RealSecure was the first IDS to make a real impact into the new commercial IDS market in the mid 1990's, and is currently the market leader. The RealSecure IDS version 6.0 and above, employs a 3 tier architecture, with a Sensor tier, Event Collector tier and Management tier. As shown in figure 2.

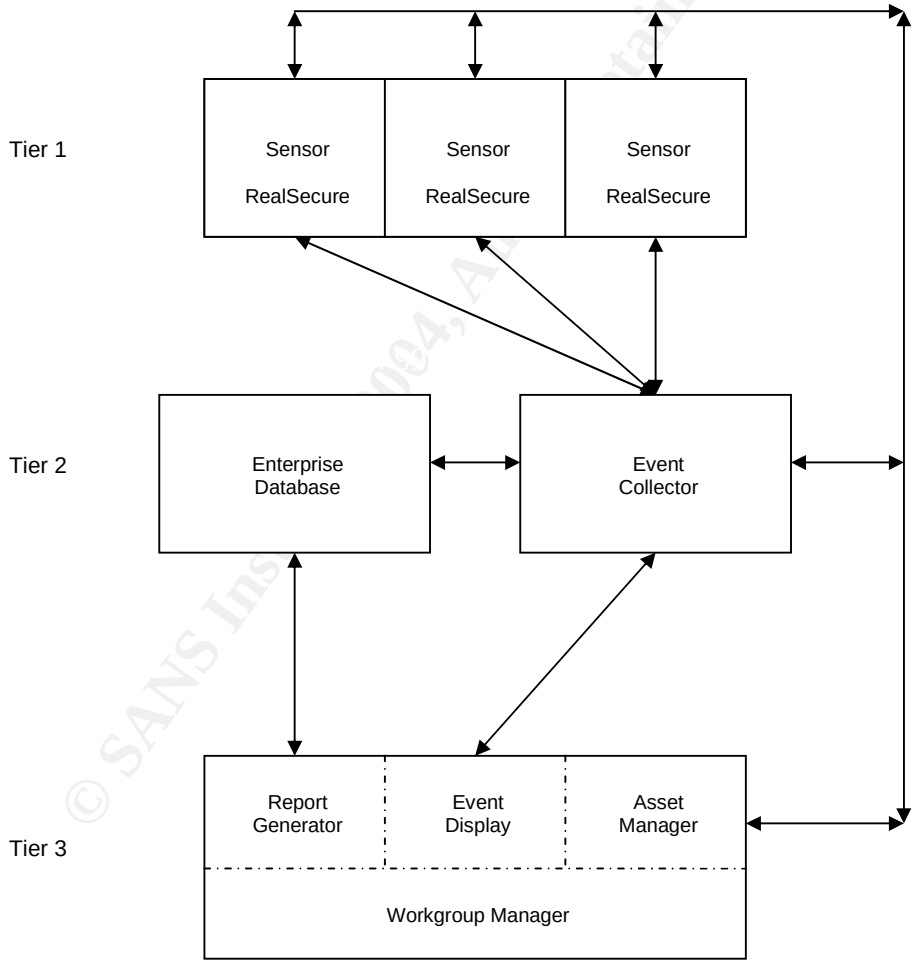


Figure 2

ISS RealSecure, 3 Tier Architecture

The Event collector is a control/synchronisation entity that is responsible for managing the flow of events into the Enterprise Database and to the event display console.

The Management tier consists of the RealSecure Workgroup Manager, which in-turn includes functionality for managing assets (sensors and event collectors), displaying events and generating reports.

When reviewing how this proprietary architecture fits into the event data management model, it can be seen that it doesn't map exactly. This isn't a problem because it's more of a modular framework, that's flexible enough to include the vast majority of IDS deployment scenarios.

Applying the event data management model to the RealSecure, 3 tier architecture, we can map the sensor tier directly to Layer 1. The event collector tier is a more difficult one to map directly, because of the event collector itself, but is most suited to layer 3. The management tier maps to layer 4 as it includes the event console within the Workgroup Manager.

The fact that there is in effect no layer 2 isn't surprising as this layer is not really required as a separate entity by a single type IDS deployment, where no custom event analysis tools are to be used. Indeed there is no need for a plugin at all in this instance. The equivalent ISS process to the layer 2 normalisation stage is event filtering. This is performed via the Workgroup Manager and involves amending the RealSecure sensor policy, which is essentially the signature set for the sensors. Therefore the user defined event filtering is performed by the sensor itself. One big advantage of treating the normalisation stage as a separate entity for a single type IDS deployment like ISS RealSecure, would be to reduce the overhead on the sensor itself. This would help to improve the performance of the sensors in coping with the high speed network traffic found today.

One key stage of event data management, where the IDS vendors, such as ISS, fail to provide sufficient functionality, is in the event handling area. This isn't surprising, given that this is more of a custom requirement, dependant partly on the underlying procedures and work practices followed by those monitoring the IDS. With this lack of vendor support, various 3rd party software developers have begun to fill the gap with Security Information Management (SIM) products such as, Tivoli Intrusion Manager, Intellitactics Network Security Manager, and NetIQ IDS Monitor and Security Manager. In fact these products are much more than event analysis consoles, and effectively provide layers 2, 3 and 4 processes, with very flexible real-time monitoring, correlation and analysis, event handling and reporting. These solutions are becoming ever more popular with MSSP's, who are not able to implement their own custom architectures or tools.

Example – MSSP using Tivoli Intrusion Manager

A managed security services provider requires a deployed architecture that allows many different security devices to be monitored for a variety of different clients. For the purposes of the following example, the use of the Tivoli Intrusion Manager product by a MSSP will be discussed with regard to the overall structure of the architecture and how it fits into the event data management model. The description of the individual components will be brief with just the main elements being covered.

The MSSP's monitored sensors' are simply layer 1 devices in the event data management model. Tivoli Intrusion Manager supports a number of IDS types as well as other security devices at layer 1, including ISS RealSecure, Cisco Secure IDS, Cisco PIX Firewall, CheckPoint Firewall-1, and others.

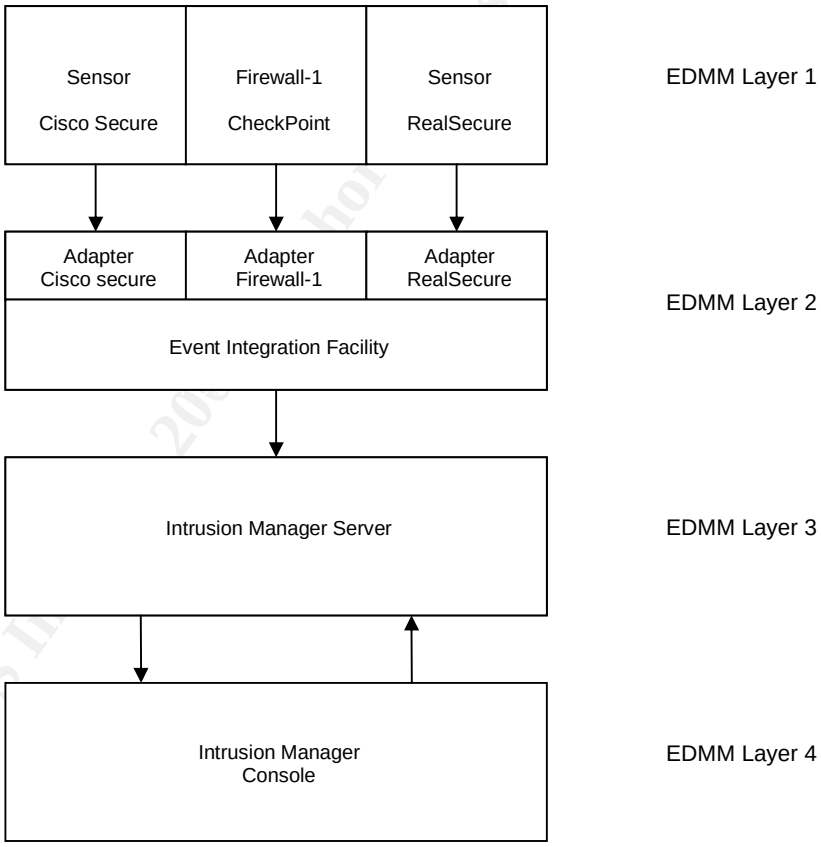


Figure 3

MSSP Tivoli Intrusion Manager Architecture

This is possible due to the Tivoli proprietary 'Adapters' that receive events from the security devices and formats them into Tivoli Intrusion Manager events. Each

different type of supported security device requires a device specific adapter to process the event data. An additional stage of processing is provided by the Event Integration Facility (EIF) which has a dual purpose. Firstly it is used to forward Tivoli Intrusion Manager events to the Tivoli Event Server, and secondly it also provides a summarisation function which condenses a large number of similar events into a much lower number of summarised events, with virtually no loss of detail. This functionality is found at layer 2 of the event data management model.

The event data passes from the EIF to the Intrusion Manager Server, which further processes the events, performs correlation and stores events in a database. This is consistent with the layer 3 processes of the event data management model.

The final layer of the Tivoli deployed architecture includes the components of the Intrusion Manager Console. The component parts are the Event Console and Crystal Reports. These provide the ability to view and handle events throughout the analysis process, and generate detailed reports on stored event data. This is the minimum Layer 4 requirement to enable practical event analysis and informative customer reporting functionality.

This type of deployment does provide the necessary architecture to enable the MSSP to support a variety of security devices, for many clients, but left simply as a delivery and display mechanism, it doesn't guarantee effective event data management.

Still too much data?

With some poetic license it could be said that the majority of effective IDS deployment architectures would fit into the event data management model. Despite the growing familiarity with IDS and adoption of new tools and services, including MSSP's, the ability to effectively handle the amount of data generated by IDS is still a major concern. It is not enough to simply adopt a deployment architecture which fits into the EDMM, it is the additional functionality of certain layer processes and the continued application of diligent tuning to the model, which differentiates between functional and effective event data management.

With the current commercially available IDS, and vast majority of Security Information Management products, the key areas where the biggest improvements in dealing with the data overload can be made are:

- Normalisation stage
- Event/Analysis Console
- Event Handling Console
- On-going tuning feedback

Normalising data at layer 2 can pay huge dividends for the person(s) who eventually have to deal with the event data. This stage can act as a very flexible filter, depending on the exact implementation. The setting of custom severity levels to events, according to a configurable set of criteria, such as IDS type, sensor, source and destination addresses and ports, event message, in any combination is a very

desirable feature. This functionality would also allow events to be dropped altogether before entering the database if they match a specific set of criteria that has been identified as a false positive. One advantage of performing this type of filtering at layer 2 is that it isn't necessary to perform this on the sensor itself, reducing the overall loading on the sensor and so helping to optimise it's ability to perform the main function of detecting suspicious behaviour.

This layer 2 function can also perform some event correlation or consolidation prior to writing to the database. For instance an ICMP:EEYE-RETINA scan may be picked up by an Enterasys Dragon sensor. The scan may be from a single source address across a whole range of network addresses, and cause many IDS events to be generated. With a process running at layer 2 that consolidates all of the same type of events from a single source, into a single IDS event of many instances, the amount of data arriving at the event console can be greatly reduced. These type of scanning events are prime candidates for this form of consolidation, and another benefit of doing it at layer 2 is that it will also reduce the number of entries in the database itself, helping to optimise the performance of the analysis console when undertaking searches.

The event console is where the analyst first sees incoming data. It's important that this interface be kept simple and free from clutter. The ability to prioritise those events that need immediate attention is very important. If via the normalisation stage at layer 2, severity levels are accurately applied, these can be used as the main criteria upon which to prioritise data in the event console display. Events of the highest severity levels can be placed uppermost or at the forefront of the display, assuring they get prompt attention. Other desirable features would include audible alarm triggers based upon event severity levels, and easy acknowledgement of events to remove them from the console display once satisfied the event poses no immediate threat.

The analysis console has an indirect impact on the effectiveness of event data management. It doesn't directly affect the number or 'quality' of events arriving at the event console, as the layer 2 processes do, but it does provide for forensic analysis of events. The results of forensic analysis should be fed back into the layer processes as on-going tuning of the EDMM to improve their efficiency, which in-turn, will over time, reduce the number, and increase the 'quality' of events arriving at the event console.

The event handling console is something that isn't really found in COTS layer 4 products. This area is more important to MSSP's and larger organisations, and provides the ability to track events as they pass through the analysis and decision making stages. One method for achieving this would be to have the ability to log the event from the event console to the event handling console. This achieves two things, firstly the event is taken out of the event console, in the same way that an acknowledgement would as discussed earlier. Secondly, from the event handling console other actions can be performed on the event such as the addition of analysis conclusions, tuning recommendations and response procedures. This can also provide a full audit trail on the event through to a satisfactory conclusion. In the case of the MSSP this is very important, as a single event may be handled by several different analysts across several shift periods. Ideally the actions that can be

performed on the events at this stage will reflect the event handling procedures employed by the organisation. This is why this functionality at layer 4 is more often reserved for custom console designs and proprietary implementations. Adding this ability to the EDMM makes for a more co-ordinated overall process, improving efficiency and audit ability.

Perhaps the most important action performed within the EDMM, is that of tuning. This includes the initial configuration of the normalisation stage, and the event filtering afforded by this process, as well as the choice of signature coverage for the sensors. This would also include the choice of audit file monitoring in the case of host sensors.

Good practice is to allow a 'settling in' period after the sensor is first deployed onto a network. This is often termed the base-lining period and will normally be associated with a great number of events being generated. At this stage the sensor will be so 'noisy' that it will be practically ineffective for monitoring purposes. It will also be the period when the most tuning attention should be applied to the sensor. The aim of this base-lining is to reduce the total number of 'poor quality' events being generated as the sensor is 'tuned' in harmony with the type of network traffic it has been deployed to monitor. With the flexibility of the layer 2 processes of the EDMM this tuning can be very effective.

Once the IDS system has been in place for a short period and the base-lining phase has been completed, the overall system will require constant tuning feedback to be applied to the layer 1 and 2 processes mostly. With additional event correlation and consolidation occurring at layer 2 there is great scope to aid the overall event data management process. The concept of tuning should not be to simply remove signatures that have generated false positives, but to use the concept of severity levels to apply an importance rating to the events, based upon the context in which the event was triggered. This is more effective if the layer 2 normalisation stage has the flexibility to accommodate a full set of event criteria, which can be combined to create what is in effect a new rule for an existing rule.

For example basic port-scanning activity can be set to a low level of severity such as level 1, and other activity scaled up to the highest severity level of 5, such as backdoor traffic. By defining an event handling strategy based around this concept of event severity levels, the sometimes vast amount of event data can be effectively categorised and handled according to the perceived importance of the event. Thus level 1 activity need not automatically be forwarded to the event console, reducing the overall amount of data that requires prompt attention. This lower level data can be accessed when required from the analysis console, in the form of visualisation tools or advanced search functionality. For this system to maintain its integrity the tuning has to be very carefully considered, and the severity levels set must still remain open for interpretation, and not be taken for granted.

Summary

Current IDS are still a useful addition to security defences, despite their inherent, perceived nature of data overload. By adopting an architecture which allows the functionality described in this paper and applying an event handling strategy that prioritises incoming events, data overload need not be a foregone conclusion. The deployed system requires constant updating to remain effective and constant tuning to be efficient. Until the next generations of IDS overcome the problems of dealing with large amounts of data, if indeed they do, then effective event data management is the only real answer.

References:

1 http://documents.iss.net/whitepapers/rs60_wgm_arch_deploy.pdf

2 http://publib.boulder.ibm.com/tividd/td/TIM/GC32-0747-01/en_US/PDF/GC32-0747-01.pdf

3 http://www.intellitactics.com

4 http://www.netiq.com/products/sm/default.asp?menu=solutions_security_incident_c_menu.xml

5 http://www.netiq.com/products/sm/secure.asp

Part 2 – Network Detects

The following section provides detailed analysis of three network detects, a, b and c. Each detect is based upon a different form of attack, and the data is taken from various sources of raw logs which have been captured 'in the wild'. The analysis of each detect is split into ten parts.

Note: all "bad checksums" should be ignored since the addresses have been changed to protect the real identity of the hosts captured in the log files.

Detect (a) - DNS named version attempt

(a-1) – Source of Trace:

This detect was taken from the raw tcpdump binary log files available at incidents.org. The url for the specific log file used is as follows:

http://www.incidents.org/logs/RAW/2002.5.5

(a-2) – Detect was generated by:

The detect was generated from the raw log files by using Snort to read the relevant file, process it and output Snort alerts as determined by the configuration settings within the 'snort.conf' file. The following command was used:

```
# snort –c /etc/snort/snort.conf –r 2002.5.5
```

Snort is a freely available Intrusion Detection System, packet sniffer and logger. Full details and downloads can be found at http://www.snort.org

For the purposes of this detect Snort v1.9.0 (Build 209) was used, running on a Red Hat 7.2 Linux system. The snortrules-stable.tar.gz file was downloaded from the snort.org site, on February 16th 2003, and used to provide the rules for attack detection.

Other tools used in the analysis of this detect included:

Analysis Console for Intrusion Detection (ACID), a PHP based front-end querying and display tool. This was used in conjunction with Snort configured to log to a MySQL database: http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html

TCPDUMP, a freely available packet sniffer and logger: http://www.tcpdump.org

Ethereal, a freely available GUI based packet sniffer and logger: http://www.ethereal.com

The alerts generated that will be concentrated upon in the following commentary were shown in the ACID console, in a similar fashion to that given below:

<Signature>	<Timestamp>	<Source Address>	<Dst Address>	<Prot>
DNS named version attempt	2002-06-05 04:55:13	203.122.47.137:20512	46.5.219.77:53	UDP
DNS named version attempt	2002-06-05 05:12:19	203.122.47.137:15546	46.5.35.141:53	UDP
DNS named version attempt	2002-06-05 05:32:11	203.122.47.137:12825	46.5.229.249:53	UDP
DNS named version attempt	2002-06-05 05:35:43	203.122.47.137:16288	46.5.223.34:53	UDP
DNS named version attempt	2002-06-05 05:41:09	203.122.47.137:21551	46.5.57.150:53	UDP
DNS named version attempt	2002-06-05 06:06:36	203.122.47.137:24203	46.5.105.27:53	UDP
DNS named version attempt	2002-06-05 07:39:20	203.122.47.137:27898	46.5.227.72:53	UDP
DNS named version attempt	2002-06-05 07:55:32	203.122.47.137:21758	46.5.101.45:53	UDP
DNS named version attempt	2002-06-05 08:32:11	203.122.47.137:13368	46.5.39.230:53	UDP
DNS named version attempt	2002-06-05 09:05:33	203.122.47.137:23652	46.5.177.16:53	UDP
DNS named version attempt	2002-06-05 10:41:28	203.122.47.137:28760	46.5.223.68:53	UDP
DNS named version attempt	2002-06-05 11:07:30	203.122.47.137:31949	46.5.250.2:53	UDP
DNS named version attempt	2002-06-05 13:31:28	203.122.47.137:14833	46.5.196.191:53	UDP

The Snort signature which detected these attacks:

alert udp $EXTERNAL_NET any -> $DNS_SERVERS 53 (msg:"DNS named version attempt"; content:"|07|version"; nocase; offset:12; content:"|04|bind"; nocase; offset: 12; reference:nessus,10028; reference:arachnids,278; classtype:attempted-recon; sid:1616; rev:4;)

Breaking this signature down requires a knowledge of the snort rule language and structure. More information on snort rules can be found at: http://packetstormsecurity.nl/papers/IDS/snort_rules.htm

Every snort rule has two main parts, the header field and the options field. The header field is the first part of the signature up to the parentheses:

alert udp $EXTERNAL_NET any -> $DNS_SERVERS 53

This can be interpreted as: generate an alert and log the packet if the protocol is UDP, from a source which is defined by the variable $EXTERNAL_NET on any port, to a destination defined by the variable $DNS_SERVERS on port 53.

However this does not mean that the header alone will trigger an alert. The options field of the signature must then be taken into account by the Snort program and only if these conditions are also met, does the signature trigger an alert. Looking more closely at the options field, we see that the text found within the quotes immediately after the 'msg' descriptor is the actual Snort IDS alert description generated by the detect.

The 'content' option specifies the packet content to be looked for by the signature. In this case the signature is looking for two sets of packet content, |07|version and |04|bind. The numbers within the "| " pipe characters are byte code binary data represented as hexadecimal values, and the text following the pipe's represent the ASCII characters to be looked for in the packet content.

The 'nocase' option specifies that the ASCII content is not case dependent. The 'offset' option is an important condition to be used in conjunction with the 'content' option. It determines where to look in the packet payload for the start of the specific payload content to be found.

The remaining options fields are references for the attack detail, a classification of the perceived intent of the attack, and identifiers and revision numbers of the Snort signature itself.

Now that we've determined the conditions required to trigger the alert from the signature itself, we can expect to see these conditions within the offending packets themselves. The following log extracts for the source host 203.122.47.137 are made up of the corresponding snort alert output in the top part and the tcpdump output of the raw log file in the bottom part separated by a blank line. Each individual alert is divided from the others by the '=+=+=+=+=+' line.

```
[**] DNS named version attempt [**]
06/05-03:55:13.704488 203.122.47.137:20512 -> 46.5.219.77:53
UDP TTL:42 TOS:0x0 ID:65306 IpLen:20 DgmLen:58
Len: 38

03:55:13.704488 203.122.47.137.20512 > 46.5.219.77.domain:  [bad udp cksum f7fa!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 42, id 65306, len 58, bad cksum 924a!)
0x0000     4500 003a ff1a 0000 2a11 924a cb7a 2f89        E..:....*..J.z/.
0x0010     2e05 db4d 5020 0035 0026 7ba5 1234 0080        ...MP..5.&{..4..
0x0020     0001 0000 0000 0000 0776 6572 7369 6f6e        .........version
0x0030     0462 696e 6400 0010 0003                       .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-04:12:19.494488 203.122.47.137:15546 -> 46.5.35.141:53
UDP TTL:46 TOS:0x0 ID:23067 IpLen:20 DgmLen:58
Len: 38

04:12:19.494488 203.122.47.137.15546 > 46.5.35.141.domain:  [bad udp cksum faf7!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 46, id 23067, len 58, bad cksum ee07!)
0x0000     4500 003a 5a1b 0000 2e11 ee07 cb7a 2f89        E..:Z........z/.
0x0010     2e05 238d 3cba 0035 0026 49c9 1234 0080        ..#.<..5.&I..4..
0x0020     0001 0000 0000 0000 0776 6572 7369 6f6e        .........version
0x0030     0462 696e 6400 0010 0003                       .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-04:32:11.214488 203.122.47.137:12825 -> 46.5.229.249:53
UDP TTL:46 TOS:0x0 ID:46482 IpLen:20 DgmLen:58
Len: 38

04:32:11.214488 203.122.47.137.12825 > 46.5.229.249.domain:  [bad udp cksum f9f9!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 46, id 46482, len 58, bad cksum ce24!)
0x0000     4500 003a b592 0000 2e11 ce24 cb7a 2f89        E..:.......$.z/.
0x0010     2e05 e5f9 3219 0035 0026 8ffe 1234 0080        ....2..5.&...4..
0x0020     0001 0000 0000 0000 0776 6572 7369 6f6e        .........version
0x0030     0462 696e 6400 0010 0003                       .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-04:35:43.314488 203.122.47.137:16288 -> 46.5.223.34:53
UDP TTL:46 TOS:0x0 ID:50633 IpLen:20 DgmLen:58
Len: 38

04:35:43.314488 203.122.47.137.16288 > 46.5.223.34.domain:  [bad udp cksum f7fa!]
```

```
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 46, id 50633, len 58, bad cksum c3c6!)
0x0000   4500 003a c5c9 0000 2e11 c3c6 cb7a 2f89        E..:.........z/.
0x0010   2e05 df22 3fa0 0035 0026 8850 1234 0080        ..."?..5.&.P.4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e        .........version
0x0030   0462 696e 6400 0010 0003                       .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-04:41:09.594488 203.122.47.137:21551 -> 46.5.57.150:53
UDP TTL:46 TOS:0x0 ID:56476 IpLen:20 DgmLen:58
Len: 38

04:41:09.594488 203.122.47.137.21551 > 46.5.57.150.domain:  [bad udp cksum faf7!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 46, id 56476, len 58, bad cksum 557d!)
0x0000   4500 003a dc9c 0000 2e11 557d cb7a 2f89        E..:......U}.z/.
0x0010   2e05 3996 542f 0035 0026 1c4b 1234 0080        ..9.T/.5.&.K.4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e        .........version
0x0030   0462 696e 6400 0010 0003                       .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-05:06:36.884488 203.122.47.137:24203 -> 46.5.105.27:53
UDP TTL:46 TOS:0x0 ID:19702 IpLen:20 DgmLen:58
Len: 38

05:06:36.884488 203.122.47.137.24203 > 46.5.105.27.domain:  [bad udp cksum f8f8!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 46, id 19702, len 58, bad cksum b4a0!)
0x0000   4500 003a 4cf6 0000 2e11 b4a0 cb7a 2f89        E..:L........z/.
0x0010   2e05 691b 5e8b 0035 0026 e16b 1234 0080        ..i.^..5.&.k.4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e        .........version
0x0030   0462 696e 6400 0010 0003                       .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-06:39:20.924488 203.122.47.137:27898 -> 46.5.227.72:53
UDP TTL:46 TOS:0x0 ID:16638 IpLen:20 DgmLen:58
Len: 38

06:39:20.924488 203.122.47.137.27898 > 46.5.227.72.domain:  [bad udp cksum f7fa!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 46, id 16638, len 58, bad cksum 446c!)
0x0000   4500 003a 40fe 0000 2e11 446c cb7a 2f89        E..:@.....Dl.z/.
0x0010   2e05 e348 6cfa 0035 0026 56d0 1234 0080        ...Hl..5.&V..4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e        .........version
0x0030   0462 696e 6400 0010 0003                       .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-06:55:32.964488 203.122.47.137:21758 -> 46.5.101.45:53
UDP TTL:46 TOS:0x0 ID:41753 IpLen:20 DgmLen:58
Len: 38

06:55:32.964488 203.122.47.137.21758 > 46.5.101.45.domain:  [bad udp cksum f8f8!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 46, id 41753, len 58, bad cksum 626b!)
0x0000   4500 003a a319 0000 2e11 626b cb7a 2f89        E..:......bk.z/.
0x0010   2e05 652d 54fe 0035 0026 eee6 1234 0080        ..e-T..5.&...4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e        .........version
0x0030   0462 696e 6400 0010 0003                       .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-07:32:11.964488 203.122.47.137:13368 -> 46.5.39.230:53
UDP TTL:46 TOS:0x0 ID:22186 IpLen:20 DgmLen:58
Len: 38

07:32:11.964488 203.122.47.137.13368 > 46.5.39.230.domain:  [bad udp cksum faf7!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
```

```
(ttl 46, id 22186, len 58, bad cksum ed1f!)
0x0000   4500 003a 56aa 0000 2e11 ed1f cb7a 2f89      E..:V........z/.
0x0010   2e05 27e6 3438 0035 0026 4df2 1234 0080      ..'.48.5.&M..4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e      .........version
0x0030   0462 696e 6400 0010 0003                     .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-08:05:33.074488 203.122.47.137:23652 -> 46.5.177.16:53
UDP TTL:46 TOS:0x0 ID:62197 IpLen:20 DgmLen:58
Len: 38

08:05:33.074488 203.122.47.137.23652 > 46.5.177.16.domain:  [bad udp cksum f7fa!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 46, id 62197, len 58, bad cksum c4ac!)
0x0000   4500 003a f2f5 0000 2e11 c4ac cb7a 2f89      E..:.........z/.
0x0010   2e05 b110 5c64 0035 0026 999e 1234 0080      ....\d.5.&...4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e      .........version
0x0030   0462 696e 6400 0010 0003                     .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-09:41:28.904488 203.122.47.137:28760 -> 46.5.223.68:53
UDP TTL:42 TOS:0x0 ID:57443 IpLen:20 DgmLen:58
Len: 38

09:41:28.904488 203.122.47.137.28760 > 46.5.223.68.domain:  [bad udp cksum f7fa!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 42, id 57443, len 58, bad cksum ad0a!)
0x0000   4500 003a e063 0000 2a11 ad0a cb7a 2f89      E..:.c..*....z/.
0x0010   2e05 df44 7058 0035 0026 5776 1234 0080      ...DpX.5.&Wv.4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e      .........version
0x0030   0462 696e 6400 0010 0003                     .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-10:07:30.324488 203.122.47.137:31949 -> 46.5.250.2:53
UDP TTL:42 TOS:0x0 ID:27629 IpLen:20 DgmLen:58
Len: 38

10:07:30.324488 203.122.47.137.31949 > 46.5.250.2.domain:  [bad udp cksum f7fa!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 42, id 27629, len 58, bad cksum 6c3!)
0x0000   4500 003a 6bed 0000 2a11 06c3 cb7a 2f89      E..:k...*....z/.
0x0010   2e05 fa02 7ccd 0035 0026 3043 1234 0080      ....|..5.&0C.4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e      .........version
0x0030   0462 696e 6400 0010 0003                     .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


[**] DNS named version attempt [**]
06/05-12:31:28.864488 203.122.47.137:14833 -> 46.5.196.191:53
UDP TTL:40 TOS:0x0 ID:53117 IpLen:20 DgmLen:58
Len: 38

12:31:28.864488 203.122.47.137.14833 > 46.5.196.191.domain:  [bad udp cksum f9f9!]
4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain]
(ttl 40, id 53117, len 58, bad cksum db73!)
0x0000   4500 003a cf7d 0000 2811 db73 cb7a 2f89      E..:.}..(..s.z/.
0x0010   2e05 c4bf 39f1 0035 0026 a960 1234 0080      ....9..5.&.`.4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e      .........version
0x0030   0462 696e 6400 0010 0003                     .bind.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

(a-3) – Probability the source address was spoofed:

The source address was almost certainly not spoofed. This attack is essentially a request for information about the version of BIND, the most popular implementation

of the Domain Name Service, if found to be running on the target system. Thus the aim is to illicit a reply containing the BIND version, and so there would be little point in spoofing the source address.

Of course it would be possible to spoof the source address and still capture the response, but it would require the attacker to have control of a node close to or on the network segment where the real host with the spoofed address resides. The attacker would need to sniff the response as it is returned to the real host. Any stateful external gateway or firewall would reject the response as it reached the real hosts' network as there would be no record of the initial request in the state table to match it to, so the response would most likely need to be sniffed at the perimeter of the network. Though I have mentioned this as a possibility, it is not very probable and I would not conclude this detect to be from a spoofed source address given the heading for this section.

(a-4) – Description of attack:

This attack is an information gathering attempt, to identify the BIND version running on Domain Name Servers of the target network. It is a UDP based query targeted at port 53. There are numerous vulnerabilities associated with various versions of BIND and these are mostly in the form of Denial of Service or Buffer Overflow vulnerabilities. The attacker is attempting to illicit a response from the targeted system in order to determine if it is running a vulnerable version of BIND, as a possible pre-cursor to a subsequent exploit.

Information on various BIND versions and associated vulnerabilities can be found at:
http://www.isc.org/products/BIND/bind-security.html

(a-5) – Attack mechanism:

By default BIND creates a zone called 'bind' in the class 'chaos'. In this zone is a TXT record (text based information) which is associated to the FQDN (Fully Qualified Domain Name) 'version.bind'. The TXT record for this host contains the BIND version.

The following excerpt is from reference:
Liu, Cricket "DNS & BIND Cookbook. October 2002
http://www.oreillynet.com/pub/a/network/excerpt/dnsbindcook_ch07/

Modern BIND name servers respond with their version to queries for TXT records attached to the pseudo domain name *version.bind* in the CHAOSNET class. For example:

```
# dig @<server-to-query> version.bind txt chaos

; <<>> DiG 9.2.1 <<>> version.bind txt chaos
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5096
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
```

```
;version.bind.                CH      TXT

;; ANSWER SECTION:
version.bind.          0      CH      TXT     "9.2.1"
```

Two tools that could have been used to cause these detects are Domain Information Groper (dig), and nslookup.

The use of dig can be seen in the above command line example. A lab generated tcpdump capture (with munged addresses) of the use of this dig command is given below:

```
23:07:22.960000  x.x.x.x.32832 >  x.x.x.x.domain:  [udp sum ok]
2043+ TXT CHAOS)? version.bind. [|domain] (DF) (ttl 64, id 0, len 58)
0x0000     4500 003a 0000 4000 4011 7e04 x  x    x  x                E..:..@.@.~.....
0x0010     x  x   x  x    8040 0035 0026 9141 0e97 0100               .j8..@.5.&.A....
0x0020     0001 0000 0000 0000 0776 6572 7369 6f6e                    .........version
0x0030     0462 696e 6400 0010 0003                                   .bind.....
```

Nslookup can be used to query DNS in much the same way as dig. The commands shown below can be used to perform a BIND version query:

```
C:\>nslookup
>server x.x.x.x
>set type=txt
>set class=chaos
>version.bind
>exit
```

A lab generated windump capture (with munged addresses) of the use of the nslookup command is given below:

```
20:06:53.599144 x.x.x.x.1514 > x.x.x.x.53:  [udp sum ok]
4+ TXT CHAOS)? version.bind. [|domain]
(ttl 128, id 18253, len 58, bad cksum 0!)
0x0000     4500 003a 474d 0000 8011 0000 x  x   x  x          E..:GM.........
0x0010     x  x    x  x   05ea 0035 0026 1abf 0004 0100        .j8....5.&.....
0x0020     0001 0000 0000 0000 0776 6572 7369 6f6e             .........version
0x0030     0462 696e 6400 0010 0003                            .bind.....
```

A comparison of the dig-tcpdump and nslookup-windump lab captures and the tcpdump logs shown in section (a-2) shows a very similar content, leading to the possibility of the use of one of these tools for this attack. The TTL values seen in the original detects (TTL = 46 in all but one detect) give clues to the probable type of attacker system being used for this attack. Default Windows systems usually set an initial TTL value of 128, whereas it is more common for 'nix varieties to set the initial TTL value to 64. Unless the packets have been through an unusually high number of hops to reach the target, the final TTL values support the probability of the source systems being of the 'nix variety.

One problem with the theory of using either dig or nslookup for this attack is the non-standard appearance of the same transaction id's being present in all of the traces captured for this analysis. Indeed all of the DNS named version attempts captured in the log file have the transaction id set to the same hex value of 1234 (decimal 4660). This does not comply with the normal random generation of transaction id's, seen in normal DNS traffic.

Consider the following lab generated dig version requests. These were made over a short period of time from a Red Hat Linux 7.2 system. In each successive trace the last octet of the target IP address was incremented by a value of 1, to mimic the choice of different targets for each of the original attacks.

15:36:47.150000 x.x.x.x.32835 > x.x.x.x.domain: [udp sum ok] 14447+ TXT CHAOS)? version.bind. [|domain] (DF) (ttl 64, id 0, len 58)

15:37:00.400000 x.x.x.x.32836 > x.x.x.x .domain: [udp sum ok] 62031+ TXT CHAOS)? version.bind. [|domain] (DF) (ttl 64, id 0, len 58)

15:37:18.830000 x.x.x.x.32836 > x.x.x.x.domain: [udp sum ok] 51996+ TXT CHAOS)? version.bind. [|domain] (DF) (ttl 64, id 0, len 58)

15:37:36.120000 x.x.x.x.32837 > x.x.x.x.domain: [udp sum ok] 28577+ TXT CHAOS)? version.bind. [|domain] (DF) (ttl 64, id 0, len 58)

15:37:54.990000 x.x.x.x.32838 > x.x.x.x.domain: [udp sum ok] 30954+ TXT CHAOS)? version.bind. [|domain] (DF) (ttl 64, id 0, len 58)

The above traces show that the transaction id (shown blue and underlined), does change and is definitely not static.

There are other tools which could have been used to query the BIND version, such as the Nessus security scanner referenced in the snort signature which generated these detects. However Nessus is more usually used to scan for multiple vulnerabilities on a target host and not in this context of a single 'vulnerability' across multiple hosts.

The other possibility to consider is that given the static transaction id's of all the traces in the log file, an automated scanner was being used to probe for DNS servers' BIND versions. This does seem to be the most probable source of the attacks. I haven't been able to track down any automated BIND version scanning tools in the time available.

(a-6) – Correlations:

The attack 'DNS named version attempt' has a corresponding arachnids reference: http://www.whitehats.com/info/IDS278

The source address 203.122.47.137, when resolved using the online tools from http://centralops.net and http://www.apnic.net is found to be registered to the Indian company Spectra Net, and appears to be from a pool of DSL addresses, leased by this company to an industrial estate in Okhla, New Delhi.

Information available from http://www.mynetwatchman.com shows that this host was active during late 2001 and throughout 2002, querying many different targets for BIND versions, in the same way as described in the above detects. MyNetwatchman attempted to contact Spectra Net in February and March 2002 with an 'abuse' email providing details of some of the activity captured by MyNetwatchman for this host. The last entry on this web site for this host was in January 2003 and the web site has now closed the incidents relating to 203.122.47.137, citing 'no recent activity' as the reason.

The http://www.incidents.org/logs/RAW/2002.5.5 log file contained no further activity from this host or any other hosts with addresses registered to Spectra Net.

However one correlation found in the log file is that all of the DNS named version attempt Snort alerts originated from addresses in the Asia Pacific region. This is a loose correlation and no real conclusions can be drawn from this information alone. When further correlation is applied, all of these Asia Pacific addresses within the log file show the same transaction id values as discussed above in section a-5. This allows many suppositions to be made about possible hacker groups or individuals in this region having access to this automated tool, but no firm conclusions can be drawn.

There have been many BIND vulnerabilities exposed in the past with some, very serious indeed. The SANS/FBI top twenty Most Critical Internet Security Vulnerabilities has consistently listed BIND and can be found at the following url: http://www.sans.org/top20/

Another good source of information about BIND vulnerabilities is: http://www.isc.org/products/BIND/bind-security.html

(a-7) – Evidence of active targeting:

When considering the active targeting of hosts, the obvious aspect of the alerts, are that the 13 Snort alerts target 13 individual hosts within the 46.5.x.x range. These addresses are munged and it is not possible to conclude whether the target address range may be an actual class A range or indeed if it is sub-netted in any way. With this lack of knowledge, the fact that the attacks are all to individual hosts suggests that the attack is more likely to be a general scan across arbitrary hosts for DNS BIND versions rather than a targeted attack against known DNS servers.

It is possible that the attacker is looking for a particular version of BIND, before attempting a preferred exploit against it. It may also be a more general scan for any vulnerable version of BIND for which the attacker will attempt an exploit.

No other entries for these hosts (either source or destination) are contained in the logs at all.

(a-8) – Severity:

Severity is calculated using the following formula:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Where a scale of values from 1=lowest, to 5=highest are used.

Criticality = 1

This is a measure of how critical the target system is. However it is not possible from the log file to determine whether any of the target systems are in fact DNS servers. If they were, criticality would be 5. Since I have stated in section a-7 that this is probably not a targeted attack against known DNS servers, the criticality value has been set to a low value.

Lethality = 2
The attack itself is really just an information gathering attempt. The information gleaned (if any) from this attack would not necessarily mean that an exploit was possible or inevitable.

System Countermeasures = 1
Nothing is really known about the target systems themselves, so it is best to err on the side of caution and give the lowest score available.

Network Countermeasures = 2
Not much is known about the defensive mechanisms of the target network, except the fact that at least one Snort sensor is being used. The use of an IDS sensor could alert to the initial probing for possible vulnerabilities (as in this case) or an actual compromise taking place. This isn't going to stop the attack directly, but can, in alerting to the probing occurring, influence the security decisions made about any BIND DNS servers deployed on the network. In the case of a compromise occurring, the IDS would allow prompt action to be taken to limit the damage caused by such an attack.

Severity of this attack is therefore (1 + 2) – (1 + 2) = 0

(a-9) – Defensive recommendations:

As this attack is an attempt to gain a reply from the DNS server about the version of BIND running on it, the simplest defensive measure to employ would be to stop the DNS server from replying with it's version altogether. This can be achieved by adding the 'version' statement to the 'options' section in the named.conf file of BIND.

BIND has a feature named ACL (Access Control Lists). This will allow specific hosts such as localhost to gain the version, but not other systems as described at the following url: http://nakedape.cc/wiki/index.cgi/BindNotes

Add the following to your named.conf file:

```
acl "trusted"   {  {127.0.0/8; };
};

zone "bind" chaos {
        type master;
        file "/var/named/bind";
        allow-query { trusted; };
        allow-transfer { none; };
};
```

Then the file /var/named/bind needs to be created:

```
TTL 1D
```

```
#ORIGIN bind.
@       1D  CHAOS SOA   localhost.      root.localhost. (
                                1
                                3H
                                1H
                                1W
                                1D  )
        CHAOS  NS       localhost.
```

This should ensure that only the localhost may view the version.

Alternatively in BIND version 8.2 and later, the system can be configured to return false information or warning messages. One way to accomplish this is to use the *version options* sub-statement found in the named.conf file. This configuration returns a user defined string upon BIND DNS named version queries:

```
options {
        directory "/var/named";
        version "What's it got to do with you";
};
```

The most obvious defensive measure against a possible exploit is to make sure that any DNS servers deployed on the network are either running the latest version of BIND (if this is the preferred choice), or running a version that is fully patched.

Systems that are not required to perform name resolution shouldn't be running BIND in the first place unless there is a very good reason, such as a lab system etc. This recommendation should certainly apply to any externally visible systems.

The latest version of the Secure BIND Template can be found at this url: http://www.cymru.com/Documents/secure-bind-template.html

(a-10) – Multiple choice test question:

When using the dig tool to query a DNS server for its BIND version which of the following statements is true:

a. dig issues an inverse query for the bind.version txt record
b. dig issues a standard query for the bind.version chaos record
c. dig issues an inverse query for the version.bind chaos record
d. dig issues a standard query for the version.bind txt record

answer is d

By default BIND creates a zone called 'bind' in the class 'chaos'. In this zone is a TXT record (text based information) which is associated to the FQDN (Fully Qualified Domain Name) 'version.bind'. The TXT record for this host contains the BIND version.

Detect (a) posting – intrusions@incidents.org:

This detect was posted to the above forum on 09/03/2003. No replies were posted up to the submission date of this paper.

Submitted by: Antony Gummery Date Submitted: 05/04/2003

Detect (b) – SCAN nmap TCP

(b-1) – Source of Trace:

This detect was taken from the raw tcpdump binary log files available at incidents.org. The url for the specific log file used is as follows:

http://www.incidents.org/logs/RAW/2002.9.17

(b-2) – Detect was generated by:

The detect was generated from the raw log files by using Snort to read the relevant file, process it and output Snort alerts as determined by the configuration settings within the 'snort.conf' file. The following command was used:

```
# snort –c /etc/snort/snort.conf –r 2002.9.17
```

Snort is a freely available Intrusion Detection System, packet sniffer and logger. Full details and downloads can be found at http://www.snort.org

For the purposes of this detect Snort v1.9.0 (Build 209) was used, running on a Red Hat 7.2 Linux system. The snortrules-stable.tar.gz file was downloaded from the snort.org site, on February 16th 2003, and used to provide the rules for attack detection.

Other tools used in the analysis of this detect included:

Analysis Console for Intrusion Detection (ACID), a PHP based front-end querying and display tool. This was used in conjunction with Snort configured to log to a MySQL database: http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html

TCPDUMP, a freely available packet sniffer and logger: http://www.tcpdump.org

Ethereal, a freely available GUI based packet sniffer and logger: http://www.ethereal.com

The alerts generated that will be concentrated upon in the following commentary were shown in the ACID console, in a similar fashion to that given below:

<Signature>	<Timestamp>	<Source Address>	<Dst Address>	<Prot>
SCAN nmap TCP	2002-10-17 17:25:45	193.144.127.9	32.245.136.215	TCP
SCAN nmap TCP	2002-10-17 17:25:50	193.144.127.9	32.245.136.215	TCP
SCAN nmap TCP	2002-10-17 17:25:55	195.77.24.2	32.245.136.215	TCP
SCAN nmap TCP	2002-10-17 17:26:00	195.77.24.2	32.245.136.215	TCP

Submitted by: Antony Gummery Date Submitted: 05/04/2003

The Snort signature which detected these attacks:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap TCP";flags:A;ack:0; reference:arachnids,28;
classtype:attempted-recon; sid:628; rev:1;)
```

Breaking this signature down requires a knowledge of the snort rule language and structure. More information on snort rules can be found at: http://packetstormsecurity.nl/papers/IDS/snort_rules.htm

Every snort rule has two main parts, the header field and the options field. The header field is the first part of the signature up to the parentheses:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
```

This can be interpreted as: generate an alert and log the packet if the protocol is TCP, from a source which is defined by the variable $EXTERNAL_NET on any port, to a destination defined by the variable $HOME_NET on any port.

However this does not mean that the header alone will trigger an alert. The options field of the signature must then be taken into account by the Snort program and only if these conditions are also met, does the signature trigger an alert. Looking more closely at the options field, we see that the text found within the quotes immediately after the 'msg' descriptor is the actual Snort IDS alert description generated by the detect.

The 'flags' field determines the flag settings to match within the offending packet. Possible TCP Flags are:

SYN:	establish a new TCP session
ACK:	acknowledge data receipt
PUSH:	send data
RESET:	abort a TCP session
FIN:	terminate a TCP session gracefully
URG:	send data urgently

In the case of the Snort signature above the flag 'A' is being looked for which corresponds to the ACK flag in the TCP packet. The 'ack' field is the value of the acknowledgement number the signature is also looking for, which in this case is set to zero.

The remaining options fields are references for the attack detail, a classification of the perceived intent of the attack, and identifiers and revision numbers of the Snort signature itself.

Now that we've determined the conditions required to trigger the alert from the signature itself, we can expect to see these conditions within the offending packets themselves. The Snort message names nmap as the tool used for this attack. This is an assumption and given what the signature is designed to detect, this assumption is based upon probability given the popularity of the nmap tool and its older versions use of the 'ack' scan method, and an acknowledgement field set to zero.

The following log extracts for the target host 32.245.136.215 are made up of the corresponding snort alert output in the top part and the tcpdump output of the raw log file in the bottom part separated by a blank line. Each individual alert is divided from the others by the '=+=+=+=+=+' line.

```
[**] SCAN nmap TCP [**]
10/17-17:25:45.266507 193.144.127.9:80 -> 32.245.136.215:137
TCP TTL:44 TOS:0x0 ID:54253 IpLen:20 DgmLen:40
***A**** Seq: 0x282  Ack: 0x0  Win: 0x578  TcpLen: 20

17:25:45.266507 193.144.127.9.http > 32.245.136.215.netbios-ns: .
[bad tcp cksum 1815!] 642:642(0) ack 0 win 1400
(ttl 44, id 54253, len 40, bad cksum bb64!)
0x0000     4500 0028 d3ed 0000 2c06 bb64 c190 7f09          E..(....,..d....
0x0010     20f5 88d7 0050 0089 0000 0282 0000 0000          .....P..........
0x0020     5010 0578 a783 0000 0000 0000 0000               P..x..........
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] SCAN nmap TCP [**]
10/17-17:25:50.266507 193.144.127.9:80 -> 32.245.136.215:137
TCP TTL:44 TOS:0x0 ID:54531 IpLen:20 DgmLen:40
***A**** Seq: 0x2E9  Ack: 0x0  Win: 0x578  TcpLen: 20

17:25:50.266507 193.144.127.9.http > 32.245.136.215.netbios-ns: .
[bad tcp cksum 1815!] 103:103(0) ack 0 win 1400
(ttl 44, id 54531, len 40, bad cksum ba4e!)
0x0000     4500 0028 d503 0000 2c06 ba4e c190 7f09          E..(....,..N....
0x0010     20f5 88d7 0050 0089 0000 02e9 0000 0000          .....P..........
0x0020     5010 0578 a71c 0000 0000 0000 0000               P..x..........
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] SCAN nmap TCP [**]
10/17-17:25:55.256507 195.77.24.2:80 -> 32.245.136.215:137
TCP TTL:50 TOS:0x0 ID:54783 IpLen:20 DgmLen:40
***A**** Seq: 0x347  Ack: 0x0  Win: 0x578  TcpLen: 20

17:25:55.256507 195.77.24.2.http > 32.245.136.215.netbios-ns: .
[bad tcp cksum 1815!] 839:839(0) ack 0 win 1400
(ttl 50, id 54783, len 40, bad cksum 189d!)
0x0000     4500 0028 d5ff 0000 3206 189d c34d 1802          E..(....2....M..
0x0010     20f5 88d7 0050 0089 0000 0347 0000 0000          .....P.....G....
0x0020     5010 0578 0c09 0000 0000 0000 0000               P..x..........
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] SCAN nmap TCP [**]
10/17-17:26:00.256507 195.77.24.2:80 -> 32.245.136.215:137
TCP TTL:50 TOS:0x0 ID:54970 IpLen:20 DgmLen:40
***A**** Seq: 0x375  Ack: 0x0  Win: 0x578  TcpLen: 20

17:26:00.256507 195.77.24.2.http > 32.245.136.215.netbios-ns: .
 [bad tcp cksum 1815!] 46:46(0) ack 0 win 1400
(ttl 50, id 54970, len 40, bad cksum 17e2!)
0x0000     4500 0028 d6ba 0000 3206 17e2 c34d 1802          E..(....2....M..
0x0010     20f5 88d7 0050 0089 0000 0375 0000 0000          .....P.....u....
0x0020     5010 0578 0bdb 0000 0000 0000 0000               P..x..........
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

(b-3) – Probability the source address was spoofed:

The source address was almost certainly not spoofed. The attack is designed to penetrate stateless firewalls and simple packet filtering gateways. The response from the target host can indicate the type of filtering present at the gateway. This can only be determined by the attacker if a judgement can be made as to whether or not a response was sent from the target host.

(b-4) – Description of attack:

The attack is essentially an information gathering attempt. This is a type of scan and can be used to ascertain if a target host is 'live' or it can also be used to determine if filtering ('firewalling') is in place between the attacker and the target and whether a firewall is stateful or just a simple packet filtering gateway that blocks incoming SYN packets. The key to this attack method is the type of response returned by the initial stimulus, and the way this can be interpreted to identify live hosts and provide clues to the possible network gateway topology.

The most common tool for generating such a scan is nmap, and in addition to the 'ack' scan older versions of this tool (up to and including v2.53-1) set the acknowledgement field to zero.

Indeed this is the main reason that the Snort signature message names this tool as the source of the attack, and therefore only alerts to the use of an older version of nmap, 'ack' scanning occurring.

http://www.insecure.org/nmap/

(b-5) – Attack mechanism:

This scan type sends an ACK packet to the target port(s) specified. The response from the target will help to determine if the host is accessible, and as a consequence can help to determine what type of filtering (if any) is present between the attacker and the target host.

The use of the 'ack' scan method does not distinguish between open or closed ports, and if a host receives the incoming 'ack' packet it will send a 'rst' (reset) packet back whether or not the port is open or closed. Therefore this scan is not an attempt to determine if a specific port is open on the target.

The following is an extract from RFC 793 describing what should happen in response to a packet arriving at a target host that is not part of an established TCP connection.

RFC 793: http://www.ietf.org/rfc/rfc0793.txt?number=793

"If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYN's addressed to a non-existent connection are rejected by this means.

If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the CLOSED state."

The response can also help determine the presence of firewalls and corresponding ACL's. If a RST comes back, the port is classified as "unfiltered". If nothing comes back (or if an ICMP unreachable is returned), the port is classified as "filtered".

For these particular attacks against the target host the attacker has most probably used the nmap tool, version prior to 2.54, to perform the scan. A source port of 80 has been used, which is has almost certainly been chosen to help evade packet filtering gateways from intercepting the attacks. A source port of 80 will normally be allowed through firewalls as this is the standard port used by HTTP web servers. The use of the 'ack' flag setting may also help to evade IDS detection, as most versions of Snort, to name but one IDS, don't detect general 'ack' scans in the same way a 'syn' scan or 'fin' scan would be detected.

The choice of target port is interesting in that it would make more sense to use an ephemeral target port (greater than 1023). Server or well known ports (less than 1024) are more likely to be blocked by gateway devices, as external hosts are generally not allowed to access services on the internal, 'protected' network. Indeed this is why the standard DMZ exists.

The lab generated logs below were captured using tcpdump. The tool nmap for windows (NMapWin v1.3.1) was used to simulate a similar attack to that being analysed in this section of the paper. For information on the usage of nmap see the man page: http://www.insecure.org/nmap/data/nmap_manpage.html

```
#nmap -sA -P0  -p 137  -g 80 -T 3 x.x.x.x

17:07:09.015909 192.168.1.8.http > 192.168.1.150.netbios-ns: .
[tcp sum ok] 146959726:146959726(0) ack 329593022 win 4096
(ttl 51, id 34893, len 40)
0x0000     4500 0028 884d 0000 3306 7b94 x   x   x   x       E..(.M..3.{.....
0x0010     x   x   x   x  0050 0089 08c2 6d6e 13a5 30be       .....P....mn..0.
0x0020     5010 1000 6079 0000 0000 0000 0000                 P...`y........

17:07:09.015909 192.168.1.150.netbios-ns > 192.168.1.8.http: R
[tcp sum ok] 329593022:329593022(0) win 0 (DF)
(ttl 255, id 0, len 40)
0x0000     4500 0028 0000 4000 ff06 f7e0   x   x   x   x       E..(..@.........
0x0010     x   x   x   x  0089 0050 13a5 30be 0000 0000       .......P..0.....
0x0020     5004 0000 e6b5 0000                                 P.......
```

This illustrates the point being made in RFC 793 as the reset (RST) packet returns a sequence number with the same value as the offending 'ack' packet.

This also shows however that this recent windows version of nmap doesn't generate 'ack' values of zero as discussed in section b-4 above. Therefore the use of this version of the tool would not result in the Snort signature specified in section b-2, being triggered.

To prove this Snort was used to read the tcpdump file of this lab generated attack and output any subsequent alerts, using the following command:

```
# snort –c /etc/snort/snort.conf –r dumpfile.log
```

As expected no alerts were generated from this log file.

In order to verify that an older version of nmap would produce the 'ack' values of zero necessary to trigger the Snort signature, version 2.53 of the tool was used as below:

#nmap –e eth1 –sA –P0 –p 137 –g 80 –t 3 x.x.x.x

The resulting packets were captured using tcpdump, and can be seen below:

```
17:44:29.855909 x.x.x.x.http > x.x.x.x.netbios-ns: .
[tcp sum ok] 1262423899:1262423899(0) ack 0 win 2048
(ttl 57, id 56915, len 40)
0x0000    4500 0028 de53 0000 3906 1f8e x  x  x  x        E..(.S..9.....
0x0010    x  x   x  x   0050 0089 4b3f 0f5b 0000 0000     .....P..K?.[....
0x0020    5010 0800 c872 0000                            P....r..

17:44:29.855909 x.x.x.x.netbios-ns > x.x.x.x.http: R
[tcp sum ok] 0:0(0) win 0 (ttl 128, id 20010, len 40)
0x0000    4500 0028 4e2a 0000 8006 68b7 x  x  x  x        E..(N*....h.....
0x0010     x  x   x  x  0089 0050 0000 0000 0000 0000     .......P........
0x0020    5004 0000 2b19 0000 0000 0000 0000             P...+.........
```

These traces prove that the older versions of nmap do indeed generate 'ack' values of zero, as discussed by the author of the nmap tool, Fyodor himself at the following url: http://archives.neohapsis.com/archives/snort/2000-08/0152.html

Incidentally, the returned sequence number of the reset packet is once again equal to the value of the offending acknowledgement number, in this case zero. This is further confirmation of the RFC 793 directive. As the 'ack' value is zero in these packets, if read by Snort, they should result in the 'SCAN nmap TCP' signature specified in section b-2, being triggered. To prove this, the following command was used:

snort –c /etc/snort/snort.conf –r dumpfile1.log

The resulting Snort alert was generated:

```
[**] SCAN nmap TCP [**]
03/15-17:50:22.405909 x.x.x.x:80 -> x.x.x.x:137
TCP TTL:43 TOS:0x0 ID:13808 IpLen:20 DgmLen:40
***A**** Seq: 0x53B586A8  Ack: 0x0  Win: 0x1000  TcpLen: 20
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

(b-6) – Correlations:

The attack 'SCAN nmap TCP' has a corresponding arachnids reference: http://www.whitehats.com/info/IDS28

The source addresses 193.144.127.9 and 195.77.24.2 which targeted the 'munged' address 32.245.136.215 when resolved using the online tools at http://centralops.net Returned the following information:

```
inetnum:      193.144.104.0 - 193.144.127.255
netname:      GVA
descr:        Red GVA De La Generalitat Valenciana
descr:        Valencia
country:      ES
admin-c:      EVM3-RIPE
tech-c:       AJC4-RIPE
```

Submitted by: Antony Gummery Date Submitted: 05/04/2003

inetnum: 195.77.24.0 - 195.77.24.255
netname: GVANET
descr: Generalitat Valenciana
descr: Internet access for Valencia State (NCC#1998103531)
country: ES
admin-c: JG1572-RIPE
tech-c: JG1572-RIPE

This is a positive correlation and probability would suggest this is no coincidence. The timing of the scans shows that each alert was captured at intervals of 5 seconds. This seems to rule out a single person using a single machine and simply dialling a different local ISP to gain a different IP address, as being the reason for two separate source addresses. The timings suggest the four alerts were generated by the same scan and this could be the case if the nmap tool was used with a decoy address to help obfuscate the real source address. A couple of observations that reduce the probability of this technique being used to explain the attacks is that it doesn't make much sense to only use 1 decoy address. Also the TTL values for the two addresses are different, further limiting the possibility of the same computer being used for the attacks.

The 2002.9.17 log file did not show any replies to any of the packets causing the Snort 'SCAN nmap TCP' alerts. This is to be expected given the fact that according to the README from the incidents.org web site, the log files only contain packets that violate the Snort rule set. If the target network has a stateful firewall these packets would be dropped anyway. This may be acceptable to the attacker as this scan may be used to determine the type of gateway filtering present on the target network.

No other source addresses which generated 'SCAN nmap TCP' Snort alerts from the 2002.9.17 log file, related to the same geographical area as the two above. The timings of the other similar alerts don't support any further correlation on this basis alone.

Other common findings in the 2002.9.17 log file are that all of the packets that generated Snort 'SCAN nmap TCP' alerts had a window size set to 1400 and relatively low sequence numbers, with all but three being below 1000 (decimal). This does point to similar tools or products being responsible for these scans, but there doesn't seem to an nmap setting for a particular value to be assigned to the window size field. The nmap tool also generates random sequence numbers when set to produce 'ack' scans, however in the lab I haven't been able to generate sequence numbers so low, even with various versions of nmap tested.

The choice of source and target ports is also of interest. The specific attacks being analysed in this paper used a source port of 80 and a target port of 137. Indeed all but one of the 41 Snort 'SCAN nmap TCP' alerts, used a source port of 80. This seems logical if the intent is to penetrate stateless firewalls and packet filtering gateways, which would normally allow incoming packets from the HTTP port 80. These types of gateway devices would normally be set to drop incoming 'syn' packets, in order to prevent external hosts from initiating connections to internal hosts. Hence the choice of the 'ack' packet which stands a much better chance of penetrating these gateways, especially if using a source port of 80.

The use of the target port 137 is a curious one. The use of the 'ack' scan method does not distinguish between open or closed ports, and if a host receives the incoming 'ack' packet it will send a 'rst' (reset) packet back whether or not the port is open or closed. Therefore this scan is not an attempt to determine if the NetBIOS naming service port, 137 is open on the target. As discussed in section b-5, it would make more sense, if this was simply a gateway detection scan, to use ephemeral target ports.

Information available from http://www.mynetwatchman.com shows that the host 193.144.127.9 was active during late 2001, throughout 2002 and up to March 2003. Their analysis points to a possible cause for some of the traffic seen from this host being due to a product called the LinkProof Proximity Probe from Radware. Indeed all of the source addresses responsible for generating the Snort 'SCAN nmap TCP' alerts, can be found to be linked to this product via the Mynetwatchman web site. The analysis does not elaborate on this and a pointer to the http://www.radware.com site for further information does not help determine if the characteristics of this product match those seen in the packet traces. I contacted Radware for this information, but did not receive a reply.

Further information available from http://www.mynetwatchman.com shows that the host 195.77.24.2 was also active during late 2001, throughout 2002 and up to March 2003. Similar entries on this web site are seen for this host as described in the above paragraph.

There is a definite pattern to the attacks, when considering the four alerts this paper concentrates on, and when considering all 41 Snort 'SCAN nmap TCP' alerts found within the 2002.9.17 log file.

There is a pattern of three different source addresses which scan the same target address. There are two alerts generated from each source address at five second intervals. Indeed the group of six alerts back to back are always five seconds apart. The following table shows this pattern of alerts:

2002-10-17 15:33:32	61.222.9.204:80	x.x.x.x:80
2002-10-17 15:33:37	61.222.9.204:80	x.x.x.x:80
2002-10-17 15:33:42	210.64.84.123:80	x.x.x.x:80
2002-10-17 15:33:47	210.64.84.123:80	x.x.x.x:80
2002-10-17 15:33:52	211.22.31.6:80	x.x.x.x:80
2002-10-17 15:33:57	211.22.31.6:80	x.x.x.x:80

This pattern is repeated four times throughout the period covered by the log file. These addresses, when resolved, all originate from Taipei. When put together with the other evidence, this points to a co-ordinated effort from these addresses, whether the attacks are 'ack' scans or a false positive caused by a product such as the LinkProof Proximity Probe from Radware.

(b-7) – Evidence of active targeting:

Despite all of the different correlations observed there still seems to be no conclusive evidence that these are anything other than 'ack' scan attempts to map the network gateway topology. The only tool I'm aware of that exhibits the behaviour of a zero 'ack' field seen within the packets, is an older version of nmap. Without further investigation into the Linkproof product, these attacks shouldn't be attributed to it. I did contact Radware for this information, but did not receive a reply.

In the absence of this information, analysis has to be done based upon the facts to hand. Therefore as far as the four alerts this paper concentrates on are concerned they are to be treated as attacks using 'ack' scans to map the network gateway topology. By the nature of these Snort alerts, the attacks are definitely targeted at the network and not simply errant packets arriving by accident.

The 'ack' scan is not looking for any particular target host or service running, and as such is not really active targeting in this sense. However what is being actively targeted is the network gateway topology itself. If an attacker can gain useful information about the perimeter defences, they are much more likely to be able to launch a specific attack against a target host or service.

(b-8) – Severity:

Severity is calculated using the following formula:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Where a scale of values from 1=lowest, to 5=highest are used.

Criticality = 5
This value is given because the attack is targeting the network gateway defences.

Lethality = 2
The scan itself is not a direct attack but reconnaissance activity as a possible pre-cursor to a more direct attack.

System Countermeasures = 1
Nothing is really known about the systems themselves inside the network, so it is best to err on the side of caution and give the lowest score available. This scan if successful, relies upon the receiving system to behave normally as far as the standard TCP/IP implementation is concerned and send a 'rst' packet back to the attacker.

Network Countermeasures = 2
Not much is known about the defensive mechanisms of the target network, except the fact that at least one Snort sensor is being used. The use of an IDS sensor could alert to the initial probing for possible vulnerabilities (as in this case) or an actual compromise taking place. This isn't going to stop the attack directly, but can, in

alerting to the probing occurring, influence the security decisions made about any gateway devices deployed on the network. If there were a stateful firewall in place at the gateway, this value would counter-balance the criticality value.

Severity of this attack is therefore (5 + 2) – (1 + 2) = 4

(b-9) – Defensive recommendations:

The most obvious defensive strategy against this attack is to have a well configured, stateful firewall at the network perimeter. Stateful firewalls keep track of established connections and if properly configured, would not allow an 'ack' packet that was not part of an established connection to penetrate the firewall.

These 'ack' packets are a very difficult dilemma for a simple packet filter to deal with. In response to these four attacks, filtering incoming port 137 would prevent the packet from entering the network. In fact it is good practice to filter all packets from outside address space (and the same address space as the network) targeted to server or well known ports (less than 1024) at the gateway. Details on packet filtering can easily be found through a general search engine query and an example returned by such a query is given below:

http://www.cert.org/security-improvement/practices/p058.html

(b-10) – Multiple choice test question:

Which of the following scanning methods would be best suited to gathering information about the filtering capabilities of network firewalls:

a. XMAS scan
b. SYN scan
c. ACK scan
d. NULL scan

answer is c.

The ACK scan is an advanced method usually used to map out firewall rule-sets. In particular, it can help determine whether a firewall is stateful or just a simple packet filter that blocks incoming SYN packets.

Detect (b) posting – intrusions@incidents.org:

This detect was posted to the above forum on 22/03/2003. The comments received and my subsequent replies are given below. The two persons with comments were Andrew Rucker Jones and Holger van Lengerich:

> Andrew wrote:
> What? The alerts You showed above were at intervals of seventeen minutes, twenty minutes, and

> three minutes. ... Oh wait. No, that's what ACID reports. Why does ACID report something different
> from Snort in its timestamp?

My reply: (original mistake amended)
Well spotted...I seem to have completely lost my senses when pasting into my posting.
The actual ACID timestamps are no different from the Snort or tcpdump outputs. The date was also October 17th 2002, so it wasn't just the times that were wrong. I'm so glad you picked this up.

> Andrew wrote:
> Yes... What guesses would You make for a source operating system? If it weren't for the timing, i
> would say that this is not enough to claim that the scans are not from the same machine. Still, that
> constant 5 second interval is SO suspicious. Can nmap randomize TTLs?

My reply:
Yes in fact nmap does randomize TTL values between around 37 and 64 on a per execution basis as described by Fyodor himself at the following posting: http://lists.insecure.org/lists/nmap-hackers/2000/Oct-Dec/0021.html
This seems to have been introduced for nmap version 2.05 as described at the following url: http://www.insecure.org/nmap/data/CHANGELOG

Thus my assumption that the same computer was less likely to have generated the attacks based on the given TTL values was not valid. Obviously the whole point of this is to make determining the OS from the source TTL values more difficult. Moreover the Windows port of nmap doesn't set the ack to zero as do the older 'nix versions.

> Andrew wrote:
> (b-10) - Multiple choice test question:
> I don't know. I think XMAS and NULL scans might be good for that, too. They would also let an
> attacker know if the firewall is capable of understanding and filtering such pathalogical packets.

> Holger wrote:
> While XMAS and NULL aren't valid in standard TCP and will trigger alerts on various security
> perimeters. On the contrary: a simple Ack is valid traffic with high propability, which doesn't look
> harmful [at the 1st glimpse]. So IMHO an ACK scan is better suited to recon stateless > perimeter
> filtering than XMAS and NULL. If NMAP had acked an serial other than 0, the rule hadn't triggered at
> all.

My reply:
Yes Holger I think you have summed it up nicely. Indeed the ack scan method is not generally detected by IDS, and so is the stealthiest option available in the question. Most popular IDS will with a standard ruleset will detect XMAS and NULL scans because the flag settings are in violation of standard tcp/ip behaviour, and thus they aren't prone to creating false positives. Perhaps I should have been more explicit with the question to avoid any sense of ambiguity, such as:

Which of the following scanning methods would be the stealthiest option to use, if attempting to gather information about the filtering capabilities of network firewalls?

a. XMAS scan
b. SYN scan
c. ACK scan
d. NULL scan

answer is c.

The ACK scan is an advanced method usually used to map out firewall rule-sets. In particular, it can help determine whether a firewall is stateful or just a simple packet filter that blocks incoming SYN packets. Most popular IDS systems with a standard ruleset will detect SYN, XMAS and NULL scans. The ACK packet is found in abundance in normal tcp/ip communications and so is not generally detected by IDS.

Submitted by: Antony Gummery Date Submitted: 05/04/2003

> Andrew wrote:
> Anthony,
> Good answers! Your justification of the multiple choice question was especially well put. Holger's
> was good, too. :) I agree with You, by the way. It was just a little late when i wrote that, and i was
> playing devil's advocate, too. Good luck with Your practical!

> Holger wrote:
> @Antony: Question for your practical:
> As stateless filtering is an issue in this context: Can you say something about the equipment, which
> was located in the network (layer 2), where the log's were captured? Which type of filtering do they
> probably support? [Hint: There have been some nice analysises on this list recently. ;-)]

My reply:
Yeah I guess you're referring to the more recent postings stating the use of a Snort entity between two Cisco devices. The inner device possibly being a Cisco router using access lists (ACL's) to perform packet filtering. This is generally stateless but with the addition of an IOS upgrade to context based access control (CBAC), existing routers can become more like stateful firewalls. There was no evidence of any returned packets from the scans in the log files, but then they only contain packets that trigger the Snort alerts don't they. This device could be a Cisco PIX firewall perhaps, which would be a stateful device.

> Holger wrote:
> Can it? IIRC Cisco PIXes are equipped with Intel-NICs, which obviously show up with Intel-MACs

My reply:
This is true Holger but see below for sample output of a pix

```
pix#show interface e0
interface ethernet0 "outside" is up, line protocol is up
  Hardware is i82559 ethernet, address is 0050.54fe.f8e4
  IP address 192.168.10.10, subnet mask 255.255.255.0
  MTU 1500 bytes, BW 100000 Kbit full duplex
       369836379 packets input, 3638117586 bytes, 6304 no buffer
       Received 109786 broadcasts, 0 runs, 0 giants
       4 input errors, 0 CRC, 0 frame, 4 overrun, 0 ignored, 0 abort
       309409498 packets output, 1353799512 bytes, 0 underruns
```

Note here that the i82559 is indeed an intel 10/100 interface, but see the assigned MAC. This resolves to a cisco address when using the online search facility at
http://standards.ieee.org/regauth/oui/index.shtml

Output as follows:

```
00-50-54   (hex)        CISCO SYSTEMS, INC.
005054     (base 16)    CISCO SYSTEMS, INC.
                        M/S SJA-2
                        170 W. TASMAN DRIVE
                        SAN JOSE CA 95134-1706
                        UNITED STATES
```

Intel must license these controller chips out to cisco who then assign their own MAC's?

Submitted by: Antony Gummery Date Submitted: 05/04/2003

Detect (c) – SCAN SYN FIN

(c-1) – Source of Trace:

This detect was taken from the raw tcpdump binary log files available at incidents.org. The url for the specific log file used is as follows:

http://www.incidents.org/logs/RAW/2002.6.8

(c-2) – Detect was generated by:

The detect was generated from the raw log files by using Snort to read the relevant file, process it and output Snort alerts as determined by the configuration settings within the 'snort.conf' file. The following command was used:

```
# snort –c /etc/snort/snort.conf –r 2002.6.8
```

Snort is a freely available Intrusion Detection System, packet sniffer and logger. Full details and downloads can be found at http://www.snort.org

For the purposes of this detect Snort v1.9.0 (Build 209) was used, running on a Red Hat 7.2 Linux system. The snortrules-stable.tar.gz file was downloaded from the snort.org site, on February 16th 2003, and used to provide the rules for attack detection.

Other tools used in the analysis of this detect included:

Analysis Console for Intrusion Detection (ACID), a PHP based front-end querying and display tool. This was used in conjunction with Snort configured to log to a MySQL database: http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html

TCPDUMP, a freely available packet sniffer and logger: http://www.tcpdump.org

Ethereal, a freely available GUI based packet sniffer and logger: http://www.ethereal.com

A breakdown of the 'SYN FIN' scans taken from the 2002.6.8 log file is given below, and shows that there were 57 attacks against different targets generated from a single source address of 62.153.209.202.

```
00:13:26.554488 62.153.209.202.ftp > 46.5.176.45.ftp: SF 2035067530:2035067530(0) win 1028
00:17:42.514488 62.153.209.202.ftp > 46.5.205.149.ftp: SF 426954892:426954892(0) win 1028
00:20:52.344488 62.153.209.202.ftp > 46.5.180.241.ftp: SF 615949754:615949754(0) win 1028
00:46:25.084488 62.153.209.202.ftp > 46.5.50.47.ftp: SF 1466091913:1466091913(0) win 1028
00:46:53.424488 62.153.209.202.ftp > 46.5.53.172.ftp: SF 469678064:469678064(0) win 1028
01:53:03.564488 62.153.209.202.ftp > 46.5.92.13.ftp: SF 739067695:739067695(0) win 1028
02:37:09.944488 62.153.209.202.ftp > 46.5.72.83.ftp: SF 254366780:254366780(0) win 1028
02:39:03.314488 62.153.209.202.ftp > 46.5.92.69.ftp: SF 2084567662:2084567662(0) win 1028
08:26:50.134488 62.153.209.202.ftp > 46.5.218.35.ftp: SF 1176280991:1176280991(0) win 1028
08:57:28.774488 62.153.209.202.ftp > 46.5.17.231.ftp: SF 231939338:231939338(0) win 1028
08:57:37.334488 62.153.209.202.ftp > 46.5.245.121.ftp: SF 2098352597:2098352597(0) win 1028
08:58:58.384488 62.153.209.202.ftp > 46.5.8.219.ftp: SF 1079222701:1079222701(0) win 1028
```

Submitted by: Antony Gummery Date Submitted: 05/04/2003

09:31:56.884488 62.153.209.202.ftp > 46.5.101.120.ftp: SF 1575633813:1575633813(0) win 1028
09:58:24.824488 62.153.209.202.ftp > 46.5.192.32.ftp: SF 736968395:736968395(0) win 1028
10:15:13.914488 62.153.209.202.ftp > 46.5.235.135.ftp: SF 2046297554:2046297554(0) win 1028
11:23:36.844488 62.153.209.202.ftp > 46.5.159.139.ftp: SF 824808229:824808229(0) win 1028
11:24:59.364488 62.153.209.202.ftp > 46.5.231.10.ftp: SF 275501980:275501980(0) win 1028
11:29:26.784488 62.153.209.202.ftp > 46.5.75.53.ftp: SF 1752906039:1752906039(0) win 1028
12:19:15.274488 62.153.209.202.ftp > 46.5.61.168.ftp: SF 1090160914:1090160914(0) win 1028
12:22:06.094488 62.153.209.202.ftp > 46.5.100.140.ftp: SF 632374075:632374075(0) win 1028
12:29:35.804488 62.153.209.202.ftp > 46.5.146.252.ftp: SF 1208729352:1208729352(0) win 1028
12:39:49.924488 62.153.209.202.ftp > 46.5.170.57.ftp: SF 1308624895:1308624895(0) win 1028
13:00:15.744488 62.153.209.202.ftp > 46.5.238.7.ftp: SF 1045816267:1045816267(0) win 1028
13:04:06.314488 62.153.209.202.ftp > 46.5.71.95.ftp: SF 1962072003:1962072003(0) win 1028
13:05:20.544488 62.153.209.202.ftp > 46.5.104.205.ftp: SF 922671222:922671222(0) win 1028
13:30:30.394488 62.153.209.202.ftp > 46.5.35.114.ftp: SF 175769298:175769298(0) win 1028
13:53:25.544488 62.153.209.202.ftp > 46.5.29.40.ftp: SF 455479300:455479300(0) win 1028
14:02:24.504488 62.153.209.202.ftp > 46.5.13.203.ftp: SF 812290724:812290724(0) win 1028
14:35:31.774488 62.153.209.202.ftp > 46.5.225.178.ftp: SF 2109643389:2109643389(0) win 1028
15:14:51.604488 62.153.209.202.ftp > 46.5.178.172.ftp: SF 552261075:552261075(0) win 1028
15:23:34.974488 62.153.209.202.ftp > 46.5.197.38.ftp: SF 1483341897:1483341897(0) win 1028
15:24:43.704488 62.153.209.202.ftp > 46.5.74.88.ftp: SF 1975754306:1975754306(0) win 1028
15:33:06.204488 62.153.209.202.ftp > 46.5.155.207.ftp: SF 1461820924:1461820924(0) win 1028
15:57:03.874488 62.153.209.202.ftp > 46.5.195.202.ftp: SF 1451458140:1451458140(0) win 1028
16:07:44.934488 62.153.209.202.ftp > 46.5.154.122.ftp: SF 551912906:551912906(0) win 1028
16:12:12.034488 62.153.209.202.ftp > 46.5.74.49.ftp: SF 2033647652:2033647652(0) win 1028
16:37:01.754488 62.153.209.202.ftp > 46.5.244.139.ftp: SF 1231032613:1231032613(0) win 1028
16:51:26.724488 62.153.209.202.ftp > 46.5.186.8.ftp: SF 1998987425:1998987425(0) win 1028
17:08:10.254488 62.153.209.202.ftp > 46.5.214.180.ftp: SF 1895593460:1895593460(0) win 1028
17:47:16.014488 62.153.209.202.ftp > 46.5.152.77.ftp: SF 307338054:307338054(0) win 1028
17:56:07.764488 62.153.209.202.ftp > 46.5.154.131.ftp: SF 1719947315:1719947315(0) win 1028
18:14:16.484488 62.153.209.202.ftp > 46.5.144.147.ftp: SF 169935511:169935511(0) win 1028
18:59:08.364488 62.153.209.202.ftp > 46.5.112.184.ftp: SF 1793562212:1793562212(0) win 1028
19:22:15.614488 62.153.209.202.ftp > 46.5.54.83.ftp: SF 1952477305:1952477305(0) win 1028
20:15:17.404488 62.153.209.202.ftp > 46.5.165.193.ftp: SF 1651856117:1651856117(0) win 1028
20:16:00.834488 62.153.209.202.ftp > 46.5.190.130.ftp: SF 1461635570:1461635570(0) win 1028
20:40:32.314488 62.153.209.202.ftp > 46.5.208.1.ftp: SF 1543166287:1543166287(0) win 1028
20:45:35.404488 62.153.209.202.ftp > 46.5.130.55.ftp: SF 1727006642:1727006642(0) win 1028
20:50:20.004488 62.153.209.202.ftp > 46.5.106.208.ftp: SF 2033501463:2033501463(0) win 1028
21:04:09.284488 62.153.209.202.ftp > 46.5.85.128.ftp: SF 91016597:91016597(0) win 1028
21:42:45.704488 62.153.209.202.ftp > 46.5.125.55.ftp: SF 1945531580:1945531580(0) win 1028
22:08:09.064488 62.153.209.202.ftp > 46.5.135.40.ftp: SF 1537943737:1537943737(0) win 1028
22:26:15.994488 62.153.209.202.ftp > 46.5.59.148.ftp: SF 1673396661:1673396661(0) win 1028
22:49:12.964488 62.153.209.202.ftp > 46.5.216.12.ftp: SF 563313002:563313002(0) win 1028
23:05:56.924488 62.153.209.202.ftp > 46.5.77.79.ftp: SF 166551505:166551505(0) win 1028
23:07:21.814488 62.153.209.202.ftp > 46.5.123.192.ftp: SF 91470969:91470969(0) win 1028
23:46:31.284488 62.153.209.202.ftp > 46.5.51.69.ftp: SF 814048591:814048591(0) win 1028

The Snort signature which detected these attacks:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN SYN FIN";flags:SF; reference:arachnids,198;
classtype:attempted-recon; sid:624; rev:1;)
```

Breaking this signature down requires a knowledge of the snort rule language and structure. More information on snort rules can be found at: http://packetstormsecurity.nl/papers/IDS/snort_rules.htm

Every snort rule has two main parts, the header field and the options field. The header field is the first part of the signature up to the parentheses:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
```

This can be interpreted as: generate an alert and log the packet if the protocol is TCP, from a source which is defined by the variable $EXTERNAL_NET on any port, to a destination defined by the variable $HOME_NET on any port.

However this does not mean that the header alone will trigger an alert. The options field of the signature must then be taken into account by the Snort program and only if these conditions are also met, does the signature trigger an alert. Looking more

closely at the options field, we see that the text found within the quotes immediately after the 'msg' descriptor is the actual Snort IDS alert description generated by the detect.

The 'flags' field determines the flag settings to match within the offending packet. Possible TCP Flags are:

SYN:	establish a new TCP session
ACK:	acknowledge data receipt
PUSH:	send data
RESET:	abort a TCP session
FIN:	terminate a TCP session gracefully
URG:	send data urgently

In the case of the Snort signature above, the flags 'SF' are being looked for which corresponds to both the SYN and FIN flags in the TCP packet.

The remaining options fields are references for the attack detail, a classification of the perceived intent of the attack, and identifiers and revision numbers of the Snort signature itself.

A sample of the Snort alerts generated from these attacks is given below. These log extracts are made up of the corresponding snort alert output in the top part and the tcpdump output of the raw log file in the bottom part separated by a blank line. Each individual alert is divided from the others by the '=+=+=+=+=+=+' line.

```
[**] SCAN SYN FIN [**]
07/08-00:13:26.554488 62.153.209.202:21 -> 46.5.176.45:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x794CAA8A  Ack: 0x7D8EA40C  Win: 0x404  TcpLen: 20

00:13:26.554488 62.153.209.202.ftp > 46.5.176.45.ftp: SF
[bad tcp cksum f7fa!] 2035067530:2035067530(0) win 1028
(ttl 30, id 39426, len 40, bad cksum 1940!)
0x0000     4500 0028 9a02 0000 1e06 1940 3e99 d1ca        E..(.......@>...
0x0010     2e05 b02d 0015 0015 794c aa8a 7d8e a40c        ...-....yL..}...
0x0020     5003 0404 7cb3 0000 0000 0000 0000             P...|.........
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] SCAN SYN FIN [**]
07/08-00:17:42.514488 62.153.209.202:21 -> 46.5.205.149:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x1972D08C  Ack: 0x1E1368A8  Win: 0x404  TcpLen: 20

00:17:42.514488 62.153.209.202.ftp > 46.5.205.149.ftp: SF
[bad tcp cksum f9f9!] 426954892:426954892(0) win 1028
(ttl 30, id 39426, len 40, bad cksum fcd5!)
0x0000     4500 0028 9a02 0000 1e06 fcd5 3e99 d1ca        E..(........>...
0x0010     2e05 cd95 0015 0015 1972 d08c 1e13 68a8        .........r....h.
0x0020     5003 0404 3501 0000 0000 0000 0000             P...5.........
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] SCAN SYN FIN [**]
07/08-00:20:52.344488 62.153.209.202:21 -> 46.5.180.241:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x24B6A5BA  Ack: 0x1B6C98C7  Win: 0x404  TcpLen: 20

00:20:52.344488 62.153.209.202.ftp > 46.5.180.241.ftp: SF
[bad tcp cksum f9f9!] 615949754:615949754(0) win 1028
(ttl 30, id 39426, len 40, bad cksum 157a!)
0x0000     4500 0028 9a02 0000 1e06 157a 3e99 d1ca        E..(.......z>...
0x0010     2e05 b4f1 0015 0015 24b6 a5ba 1b6c 98c7        ........$....l..
0x0020     5003 0404 3fbb 0000 0000 0000 0000             P...?.........
```

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] SCAN SYN FIN [**]
07/08-00:46:25.084488 62.153.209.202:21 -> 46.5.50.47:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x5762C989  Ack: 0x7AC02F7D  Win: 0x404  TcpLen: 20

00:46:25.084488 62.153.209.202.ftp > 46.5.50.47.ftp: SF
[bad tcp cksum f8f8!] 1466091913:1466091913(0) win 1028
(ttl 30, id 39426, len 40, bad cksum 993d!)
0x0000     4500 0028 9a02 0000 1e06 993d 3e99 d1ca        E..(.......=>...
0x0010     2e05 322f 0015 0015 5762 c989 7ac0 2f7d        ..2/....Wb..z./}
0x0020     5003 0404 76f9 0000 0000 0000 0000             P...v.........
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] SCAN SYN FIN [**]
07/08-00:46:53.424488 62.153.209.202:21 -> 46.5.53.172:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x1BFEB7F0  Ack: 0x3841907  Win: 0x404  TcpLen: 20

00:46:53.424488 62.153.209.202.ftp > 46.5.53.172.ftp: SF
[bad tcp cksum faf7!] 469678064:469678064(0) win 1028
(ttl 30, id 39426, len 40, bad cksum 96be!)
0x0000     4500 0028 9a02 0000 1e06 96be 3e99 d1ca        E..(........>...
0x0010     2e05 35ac 0015 0015 1bfe b7f0 0384 1907        ..5.............
0x0020     5003 0404 4f2a 0000 0000 0000 0000             P...O*........
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

(c-3) – Probability the source address was spoofed:

The source was almost certainly not spoofed. The attack is designed to illicit a response from the target host in order to ascertain whether or not the host is live or a specific port is open or closed. Therefore the source needs to receive the response from the target.

(c-4) – Description of attack:

The attack is an information gathering attempt. The attacker sends a packet with both the SYN and the FIN flags set. This is not a normal TCP/IP combination of flags and the response from the target host can be used to determine if the target is live and if a specific port on the target is open or closed. This technique can also be used to determine the method of filtering on the network as this flag combination can penetrate stateless firewalls and packet filtering gateways.

(c-5) – Attack mechanism:

The SIN/FIN scan works in a very similar way to the SYN scan. If the SY/FIN combination packet reaches the target, and the target port is in an open state, the target host will return a SYN/ACK packet. The source host, on receipt of this SYN/ACK, will then immediately send back a RST (reset) packet to the target host to tear down the connection. If the target port is closed, the target host will respond with a RST packet to the source.

This scan technique has both similarities and differences to the 'ACK' scanning method. The similarity is that the SYN/FIN scan can penetrate stateless firewalls and packet filtering gateways that have been configured to block incoming packets with

only the SYN flag set. The difference is that it is capable of determining whether the target port is open or closed because of the different responses generated.

Various tools can be used to generate such packets, with the two most popular being, later versions of nmap and hping2. Further information on these tools can be found at the following url's:

http://www.insecure.org/nmap/index.html
http://www.hping.org/

The interesting thing about nmap is that the functionality to produce a SYN/FIN combination packet is supposedly 'un-documented'. However a simple search on the internet yielded the fact that there was indeed a switch, not mentioned in the man pages, which allowed a SYN/FIN packet to be generated. This switch is '--synflags'.

nmap –sS --synflags SYNFIN targethost

For this particular attack, the source host is using a source port of 21 (ftp-control), and a target port of 21. The choice of this source port is most probably because it is often found to be allowed through perimeter firewalls, and thus the packet is more likely to find its' way to the target host. The choice of target port is again the ftp-control port 21, and this is most probably the real target of the scan. If the attacker was merely scanning for live hosts or mapping the gateway topology, it would make more sense to choose a high or ephemeral port (greater than 1023) as the target. This is because well known or server ports (less than 1024) are often blocked from penetrating firewalls, as a standard good practice.

For information on potential problems with the FTP protocol:
http://www.windowsecurity.com/whitepapers/Problems_With_The_FTP_PORT_Command_.html

For information on some of the FTP exploits available:
http://www.iss.net/security_center/advice/Exploits/Services/FTP/default.htm

(c-6) – Correlations:

The attack 'SCAN SYN FIN' has a corresponding arachnids reference:
http://www.whitehats.com/info/IDS198

There is also a snort signature identifier:
http://www.snort.org/snort-db/sid.html?id=624

The single source address responsible for the 57 'SCAN SYN FIN' alerts is 62.153.209.202. A scan tool such as nmap or hping2 was most probably used as described in section c-5. Looking more closely at the 57 alerts generated, highlights the fact that each packet also has the same IP identification number, which is further evidence of crafted packets being responsible for these attacks. When resolved using the online tools at http://centralops.net the following information was returned:

canonical name:mail.21-grad.de.

aliases
mail.bergkemper.com
mail.boxer-von-der-monarchie.de
mail.gesund-ernaehren-ev.de
mail.guelleruehrwerke.com
mail.guelleruehrwerke.de
mail.hamminkeln-ruft.de
mail.hhofstra.com
mail.linukz.de
mail.maschinenbauzentrum.de
mail.raumausstattungszentrum.de
mail.schwingungsdaempfer.com
mail.stahlschornsteine.com
mail.sv-suderwick.de
mail.zerspanungszentrum.de

inetnum: 62.153.209.200 - 62.153.209.207
netname: BERGKEMPER-NET
descr: Ursula Bergkemper EDV-Engineering
country: DE
admin-c: WB6989-RIPE
tech-c: WB6989-RIPE
status: ASSIGNED PA
notify: auftrag@nic.telekom.de
mnt-by: DTAG-NIC
changed: auftrag@nic.telekom.de 20010129
source: RIPE

route: 62.153.0.0/16
descr: Deutsche Telekom AG, Internet service provider
origin: AS3320
mnt-by: DTAG-RR
changed: bp@nic.dtag.de 20000207
source: RIPE

The following information was also available from centralops.net by checking the service scan option when performing a whois query:

FTP - 21: Error: Timed out

SMTP – 25 220 web.bergkemper.com ESMTP Sendmail 8.12.3/8.12.2/SuSE Linux 0.6; Wed, 19 Mar 2003 00:09:19 +0100

HTTP – 80:HTTP/1.1 200 OK Date: Tue, 18 Mar 2003 23:09:23 GMT Server: Apache/1.3.23 (Unix) PHP/4.1.0 mod_perl/1.26 Connection: close Content-Type: text/html

This host appears to be a busy system both handling mail for different domains and acting as a web server for multiple domains. Given the activity seen from this source address and the fact that this activity is not thought to be spoofed, it is possible that this system may have been compromised at some time in the past. The service scan detailed above shows that the system is currently running a version of Sendmail which is known to be susceptible to a buffer overflow, if not properly patched. Further information on the most recent Sendmail vulnerability can be found at the following url's:

CERT Advisory:
http://www.cert.org/advisories/CA-2003-07.html

Sendmail Advisory:
http://www.sendmail.org/8.12.8.html

However the dates within the log files suggest that this activity occurred in July 2002, and so this recent Sendmail vulnerability would not have applied to the system at that time. Further information available from http://www.mynetwatchman.com shows that the host 62.153.209.202 was active during July 2002 and was logged for the same attack criteria as described by this detect, and on the same date as given in the 2002.6.8 log file for the attacks. Putting all this together it is quite possible that this host was once compromised and may have since been cleaned and patched against any exploited vulnerability around the time of July 2002.

No other activity for this source was observed in the 2002.6.8 log file. The traces in the log file seems to suggest that an FTP server may be present on the target network with the 'munged' address of 46.5.180.133, which is seen in the log file as receiving 36 FTP user anonymous events from 9 different sources.

(c-7) – Evidence of active targeting:

The fact that the same source address targeted 57 different hosts on the 'munged' network could be described as active targeting. But the target was probably the FTP port 21, rather than the host systems themselves. If the scan was an attempt to gather information about the network 'firewalling', then this really is active targeting.

(c-8) – Severity:

Severity is calculated using the following formula:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Where a scale of values from 1=lowest, to 5=highest are used.

Criticality = 1
This is a measure of how critical the target system is. However it is not possible from the log file to determine whether any of the target systems are in fact FTP servers. If they were, criticality would be 3-4. Indeed if any system was to be identified as likely FTP server on the target network, it would be the host 46.5.180.133.

Lethality = 2
The scan itself is not a direct attack but reconnaissance activity as a possible pre-cursor to a more direct attack.

System Countermeasures = 1

Nothing is really known about the systems themselves inside the network, so it is best to err on the side of caution and give the lowest score available. This scan if successful, relies upon the receiving system to behave normally as far as the standard TCP/IP implementation is concerned and send a SYN/ACK back if the target port is open and a 'RST' packet back to the attacker if the port is closed. Of course this is dependent on whether the gateway devices filter the attacks or not.

Network Countermeasures = 2
Not much is known about the defensive mechanisms of the target network, except the fact that at least one Snort sensor is being used. The use of an IDS sensor could alert to the initial probing for possible vulnerable system applications (as in this case) or an actual compromise taking place. This isn't going to stop the attack directly, but can, in alerting to the probing occurring, influence the security decisions made about any gateway devices deployed on the network

Severity of this attack is therefore (1 + 2) – (1 + 2) = 0

(c-9) – Defensive recommendations:

This particular attack uses an illegal TCP flag combination of SYN/FIN in the offending packets. This can be filtered out at the gateway, without impacting on normal network operation.

As I've stated in section c-5 that the scan may be targeting either, the gateway or FTP-control port 21, then as well as the recommendation above, steps should be taken to filter traffic destined for port 21. If FTP needs to be offered as a service to systems on the target network it should be protected behind the gateway and only available to the internal systems. If FTP needs to be offered as a service to external hosts, it should be placed into the DMZ and both the FTP application and the underlying system OS needs to be well maintained, with up to date patching, and any necessary hardening applied.

There are plenty of good resources on the internet for information on securing FTP services. Just a small sample of these is given below:

http://www.cert.org/tech_tips/anonymous_ftp_abuses.html
http://community.roxen.com/developers/idocs/rfc/rfc2577.html
http://www.redhat.com/docs/manuals/linux/RHL-8.0-Manual/security-guide/s1-server-ftp.html
http://linux.omnipotent.net/article.php?article_id=3548
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windowsserver2003/proddocs/server/wsa_ftp_secure.asp

(c-10) – Multiple choice test question:

When examining a TCP header and starting from byte 0 and working towards byte 20, what is the correct order of the TCP flags:

a. URG, SYN, ACK, FIN, PSH, RST
b. SYN, ACK, PSH, URG, FIN, RST
c. URG, ACK, PSH, RST, SYN, FIN
d. SYN, PSH, URG, ACK, RST, FIN

TCP flags are found in the 13th byte offset of the TCP header. During normal host to host communication, these flags are required to inform the receiving host of the sending host's intentions.

Part 3 – Analyze This

GIAC University Security Audit

Executive Summary

This Security Audit report by Gummery Information Security Services (GISS) covers a six day period in total between 14/03/2003 to 19/03/2003.

The GIAC University provided GISS with three different types of Snort Intrusion Detection System (IDS) log files as described in the section 'Analysis Files' and tasked GISS to perform in depth analysis on these log files in an attempt to identify any security issues evident within the University campus network.

The overriding recommendation by GISS derived from the analysis, is to immediately investigate systems that have been identified as possible compromised hosts. These are detailed in the section 'Anomalous Activity'.

There was also much evidence of the following communication services operating on the University network:

- Peer to peer file sharing – Confirmed = Kazaa; Gnutella
  Suspected = GNUnet; WinMX
- Internet Relay Chat - Strongly suspected
- AOL Instant messenger - Strongly suspected

These services have known security issues and if the University has a Security Policy, these issues should be addressed in this document and strictly adhered to and enforced on the desktop.

Some other network services have been identified as a possible security concern and these should be reviewed to determine the integrity and configuration of the systems from a security standpoint. These are

- POP3 (email client access to server) accessible to external systems
- Trivial file transfer protocol (TFTP) accessible to external systems

If the University would like to discuss further the security issues raised in this report, GISS would welcome the opportunity to associate further with GIAC University.

Report compiled 05/04/2003 by Antony Gummery – Senior Security Consultant GISS

Submitted by: Antony Gummery Date Submitted: 05/04/2003

Analysis Files

The log files submitted to me by the University covered five consecutive days worth of activity on their network. The logs were generated from a Snort IDS system of an unspecified version and with a 'fairly standard rule base'.

There were three different types of logs provided by the University, scans, alert and out of spec. The out of spec log files contain packets that are inconsistent with normal TCP/IP standards. One log file of each type was provided for each of the five consecutive days:

Table - 1

Scans		
Log File	Log File Date	Log File Size KB
scans.030315.gz	15th March 2003	489
scans.030316.gz	16th March 2003	349
scans.030317.gz	17th March 2003	202
scans.030318.gz	18th March 2003	289
scans.030319.gz	19th March 2003	226

Table - 2

Alerts		
Log File	Log File Date	Log File Size KB
alert.030315.gz	15th March 2003	2067
alert.030316.gz	16th March 2003	711
alert.030317.gz	17th March 2003	1368
alert.030318.gz	18th March 2003	744
alert.030319.gz	19th March 2003	815

Table - 3

Out of Spec		
Log File	Log File Date	Log File Size KB
OOS_Report_2003_03_15_26225	14th March 2003	529
OOS_Report_2003_03_16_10675	15th March 2003	831
OOS_Report_2003_03_17_27088	16th March 2003	566
OOS_Report_2003_03_18_28243	17th March 2003	491
OOS_Report_2003_03_19_8418	18/19th March 2003	656

Once the above log files were concatenated and analysed the following table summarises the total number of entries contained within each file type:

Table - 4

Concatenated Log Files	Total entries
scans_file	177569
alert_file	477713
oos_file	7154

N.B. the alert_file contained 56,875 spp_portscan entries

The alert_file log file seemed to contain some mangled data when parsed using custom scripts to perform further analysis. Where this was evident I used standard 'cat' and 'grep' commands to extract data from the original log file and check its' content. Though this helped to overcome some of the shortcomings of the incorrectly parsed data (due to the log file format), there was some data that could not be reconciled.

The Out of Spec log file names were assumed to represent the dates that the log data was collected, however as can be seen from table 3 this was not exactly the case. There were some events dated the 19th March 2003, and although few in number, does meet the technical requirements of five consecutive days log files.

Also to maintain the confidentiality of the University's address space, and for the purposes of SnortSnarf analysis, all of the University addresses were changed from MY.NET.x.x to 999.888.x.x. This is described further in the 'Analysis Process' section at the end of Part 3.

Relational Analysis

The following analysis is mostly focused on the top talkers section of this report, and has been broken down into the different log file types provided by the University.

Scans file analysis

999.888.70.176

The top source address of 999.888.70.176 generated 9910 UDP scans, which was also the top scan entry. Analysis shows that this host may well be using the file sharing program called WinMx, which operates over UDP port 6257. This host was the source for 4919 different target addresses with the log entries having a format similar to that given below:

Mar 19 00:52:24 999.888.70.176:6257 -> 61.193.29.237:6257 UDP

Two scan entries for this University host as a target shows that it may also be using Napster as an MP3 file sharing program or WinMX on port 6699.

999.888.196.179

The second top source host in table 8 is 999.888.196.179. This host exhibits similar UDP traffic to that of the aforementioned top source, but with a different source/destination port of 22321. The exact nature of the use of this port is unclear and no conclusive information has been forthcoming on this, except to say that it is very likely peer to peer traffic that is being observed. This traffic was also observed for the three source hosts 999.888.88.134; 999.888.88.180 and 999.888.168.82.

999.888.1.3

The source host 999.888.1.3 is almost certainly a DNS server given the type of scan traffic observed to destination port 53 on many different hosts as shown below:

Mar 15 00:00:26 999.888.1.3:32807 -> 63.241.73.214:53 UDP

One interesting aspect of the entries from this host is that all of the apparent dns queries are from the same source port of 32807. This is the reason these entries are in the scans file perhaps.

The destination host 205.231.29.244 when resolved, appears to be a dns server, which from the log entries appears to be receiving queries from the previously identified University dns server 999.888.1.3.

80.60.247.181
The top destination address as given in table 14 was found to have been actively targeted by the source host 80.60.247.181 throughout the 15th March 2003 and generated 1188 entries in the scan log file. The entries were mostly NULL scans but contained numerous other strange flag combinations.

999.888.239.202
The three destination hosts from table 14 of 24.159.70.120; 62.79.69.54 and 12.221.37.190 are all recipients of packets with some strange flag combinations and NULL scans. The source of all of this activity is the University host 999.888.239.202 and as the precise nature of the traffic cannot be ascertained from the log data alone, this host should be investigated more closely.

999.888.249.194
The destination host 999.888.249.194 was the recipient of many FIN packets from different sources to target port 1214, which has been identified in the remainder of this report as being a port involved in peer to peer file sharing activities between University and external hosts.

999.888.195.67 and 999.888.202.214
The same can be said for University hosts 999.888.195.67 and 999.888.202.214 which were found to be involved in peer to peer activities on port 6346, which also discussed further in this report.

Alert file analysis

80.60.247.181
This host was chosen as it was number 6 in the top ten listing for alert_file source addresses. This host was observed generating many spp_portscan and other nmap and queso fingerprint alerts, and seemed particularly interested in the University host 999.888.234.54 as a target for this activity.

[**] NIMDA - Attempt to execute cmd from campus host [**]
999.888.195.157; 999.888.97.222; 999.888.97.72; 999.888.97.43;
These hosts were identified by this custom signature, as being probable compromised hosts by the nimda virus/worm, and attempting to propagate through exploitation of external targets.

[**] Possible trojan server activity [**]
The host **208.196.247.133** appears to have scanned a small number of target hosts within a small subset of University address space (999.888.136.x and 999.888.137.x)

for the presence of the Subseven Trojan program operating on the programs default port 27374. There is evidence in the log file to suggest that at least three University systems have been compromised by the Subseven Trojan program:

999.888.137.1:27374 > 208.196.247.133
999.888.137.17:27374 > 208.196.247.133
999.888.137.33:27374 > 208.196.247.133

University Web Servers
The following hosts appear to be University web servers identified from the alert_file, as having a source port of 80, or other criteria (such as custom signature).

Table – 5

University Web servers from alert_file				
999.888.30.3	999.888.30.4	999.888.6.7	999.888.5.20	999.888.24.34
999.888.24.44	999.888.29.3	999.888.29.66	999.888.100.165	999.888.113.208
999.888.145.18	999.888.179.77	999.888.217.206	999.888.252.133	

University TFTP Servers
The following hosts appear to be University Trivial File Transfer Protocol (TFTP) servers visible to the outside world, as seen in the alert_file with University sources communicating via source port 69:

Table – 6

University TFTP servers from alert_file				
999.888.70.146	999.888.70.172	999.888.70.218	999.888.70.232	999.888.71.88
999.888.132.1	999.888.190.1			

University NNTP Servers
999.888.24.8
This host appears to be a University NNTP server, due to the [**] EXPLOIT x86 stealth noop [**] events being observed targeted at this hosts port of 119. This signature is one of a number common to false positives for FTP/Web and NNTP traffic.

University FTP Servers
999.888.24.47 and 999.888.240.18
These hosts may well be FTP servers. This is not conclusive, as no source activity was observed from these hosts on port 21, but they were targets for this port, and each alerted with a different signature. In particular the target 999.888.24.47 looks the most likely of the two to be an FTP server due to different hosts generating the [**] FTP passwd attempt [**] when targeting this host on port 21. The other host 999.888.240.18 was the target of [**] FTP DoS ftpd globbing [**] events from a single source, this is also not a 'two way' process and is less conclusive evidence of this host being an FTP server.

University POP3 Servers
999.888.6.7; 999.888.12.4 and 999.888.25.21
The above hosts were the only University hosts observed as targets for port 110 activity, and from only 2 different sources.

[**] High port 65535 udp - possible Red Worm – traffic [**]
This signature as observed in the alert_file seems to have triggered almost total false positives with many different instances of University hosts on port 6257 communicating with external hosts on port 65535 via UDP. This is probably due to the use of a peer to peer file sharing program called GNUnet.
http://www.gnu.org/software/GNUnet/

[**] IRC evil - running XDCC [**]
This signature alerts to the usage of the IRC XDCC server which offers files in "DCC Packets" that you can request to download. A number of different University hosts used a selection of known IRC servers for this purpose and the alert_file showed these as connections to the external IRC servers on ports 6665; 6667 and 7000 mostly. A list of these University hosts and the IP addresses of known IRC servers to the canonical name given during a host lookup, is given below:

Table – 7

University hosts using IRC XDCC servers				
999.888.80.209	999.888.83.3	999.888.83.205	999.888.87.87	999.888.88.163
999.888.112.30	999.888.112.199	999.888.114.11	999.888.122.106	999.888.150.179
999.888.194.125	999.888.198.221	999.888.202.222	999.888.203.210	999.888.211.6
999.888.211.98	999.888.212.158	999.888.221.106	999.888.223.214	999.888.234.30
999.888.240.234	999.888.241.214	999.888.249.246	999.888.249.254	999.888.253.42

Table – 8

Resolved IRC XDCC servers being used by University hosts				
65.57.64.224	66.150.99.99	140.99.102.3	160.94.151.137	192.116.253.10
193.163.220.3	195.159.0.90	198.163.214.2	205.188.149.12	209.126.200.186
216.65.55.31				

999.888.30.3
This host has already been identified as a HTTP server, but also, because of the target port 524 traffic highlighted by the custom signature '999.888.30.3 activity' found in the alert_file, it appears that this host is possibly a Novell Netware system running a HTTP server.

999.888.30.4
Given what was observed for the host above, the host 999.888.30.4 is most probably running an Apache Web Server under Netware. This deduction was made because the traffic observed by the triggering of the custom signature '999.888.30.4 activity', was targeted at port 51443, which is the default port Apache configures HTTPS to use when also running Netware Enterprise Server.

999.888.3.54 and 999.888.3.56
These hosts are seen in the alert_file because of a custom signature 'Notify Brian B. 3.5x tcp' (where x= 4 or 6). There are only 23 alerts between the two hosts and it would seem that given the target port traffic observed for these systems, they are probably Microsoft based (target ports 445 and 135), and given the custom signatures, they may be experimental servers deployed on the network or perhaps there may be some concern over the well being of these systems.

[**] SUNRPC highport access! [**]

This signature can alerts to TCP or UDP packets destined to university hosts with a target port of 32771 (one signature for each protocol). This port is associated with the Sun RPC portmapper service. Normally, the rpcbind service only listens on port 111. Under Solaris, the rpcbind service also listens under port 32771, which sometimes allows attackers to bypass packet filtering. This is discussed further at the following url:
http://sunsolve.sun.com/pub-cgi/retrieve.pl?doctype=coll&doc=secbull/142&type=0&nav=sec.sba

There is a false positive observed for this signature in the alert_file, and is the result of the choice of arbitrary high port the University NNTP server is using to connect to an external NNTP server haven.net.umd.edu:
128.8.5.30:119 > 999.888.24.8:32771

There are other false positives observed for a similar reason to the above, but this time the source hosts are web servers (HTTP and HTTPS), checked using online scan tools at http://centralops.net/co/:

Table – 9

Returned web page false positives for [**]SUNRPC highport access![**]	
63.111.66.11:80 > 999.888.203.98:32771	**64.4.44.7:80 > 999.888.55.92:32771**
64.12.151.211:80 > 999.888.194.131	**64.12.174.249:80 > 999.888.203.98:32771**
64.58.76.98:80 > 999.888.145.213:32771	**64.58.76.252:80 > 999.888.206.54:32771**
65.54.194.119:80 > 999.888.252.78:32771	**65.206.229.16:443 > 999.888.110.35:32771**
66.35.250.150:80 > 999.888.203.98:32771	**66.35.250.209:80 > 999.888.70.234:32771**
66.179.48.110:80 > 999.888.252.78:32771	**66.187.232.56:80 > 999.888.149.24:32771**
66.187.232.100:443 > 999.888.84.186:32771	**128.4.40.10:80 > 999.888.55.87:32771**
128.121.26.136:80 > 999.888.204.94:32771	**128.193.0.3:80 > 999.888.223.86:32771**
152.2.210.121:80 > 999.888.83.61:32771	**171.159.65.173:80 > 999.888.149.17:32771**
192.94.38.41:80 > 999.888.100.61:32771	**194.109.137.218:80 > 999.888.83.66:32771**
204.152.189.116:80 > 999.888. 236.218:32771	**204.249.232.243:80 > 999.888.252.78:32771**
208.254.79.11:80 > 999.888.84.231:32771	**216.109.125.64:80 > 999.888.99.11:32771**

There was also another probable false positive from what appears to be a telnet session occurring between the following university host and the external host atalantis.ug.bcc.bilkent.edu.tr.
139.179.11.11:23 > 999.888.112.201:32771

The following external host was seen to generate this signature from a source port of 8000 to the target port on several University hosts of 32771. This could be one of at least two possibilities. Firstly it could be a proxy operating on port 8000, and University hosts are using this system for some reason. Secondly it may be providing streaming media of some description to the University hosts, an example follows:
193.201.220.90:8000 > 999.888.55.11:32771

Another probable false positive for this signature is the use of either ICQ or AOL Instant Messenger (AIM) from several University hosts, to different external AOL systems on port 5190 as exampled below:
64.12.29.106:5190 > 999.888.227.142:32771

[**]High port 65535 tcp - possible Red Worm – traffic[**]
There are some probable false positives observed for this signature which involves University hosts using peer to peer programs such as Kazaa, to share files with

external hosts. The following host combinations seen in the alert_file are suspected as confirming this analysis:
999.888.208.90:1214 > 64.126.78.30:65535
999.888.217.162:1214 > 151.202.169.194:65535

Out of Spec file analysis

From the oos_file many packets were seen with the reserved bits set. At first glance these packets were very similar to those generated by the Queso scanning tool http://www.securityfocus.com/tools/144. However after resolving many of the source addresses of these packets it was determined that the reserved bits were most probably set due to the use of Explicit Congestion Notification. This very subject and the false positive problem for IDS systems is discussed at the following url: http://www.sans.org/y2k/ecn.htm.

When reviewing the source addresses of the oos_file in relation to the above, it was noticed that a number of the systems were found to be legitimate SMTP servers sending SYN packets to numerous hosts on the University network via target port 25. One range of source addresses in particular were predominant in the oos_file and can be seen summarised in Table ?? below. These addresses were from the range 216.95.201.x and resolved to SMTP servers from jsuati.com. This address space and domain is a known spammer and is listed on several internet resources of such spammer lists:
http://spamcop.net/w3m?action=checkblock&ip=216.95.201.41
http://users.binary.net/dturley/procmail/spammers.txt.

There were also other identified UCE domains listed as source addresses in the oos_file. Given that these were actual SMTP servers potentially sending unsolicited commercial email (UCE or SPAM), there is a high probability that the targets of these packets seen in the oos_file are themselves SMTP servers on the University's network. A list of these is given below. The first thirteen hosts (read top left to bottom right) are the recipients of the jsuati.com packets, and the rest are added to this table as they are the recipients of packets from actual SMTP servers as found from the service scan functionality available at: http://centralops.net/co/

Table - 10

University SMTP servers from oos_file			
999.888.6.7	999.888.6.40	999.888.6.47	999.888.12.2
999.888.24.21	999.888.24.22	999.888.24.23	999.888.60.17
999.888.100.230	999.888.110.150	999.888.145.9	999.888.179.77
999.888.60.11	999.888.75.3	999.888.139.230	999.888.6.34
999.888.6.35	999.888.162.36		

N.B. the host 999.888.12.2 was added to this list as the oos_file contained four entries with this host as the source of reset packets from port 25. This host was the only SMTP source from the University in the entire file.

From Table 16 we can see that a number of target ports in the top ten listing are well known peer to peer file sharing ports. These are 6346 and 4662 and are seen in the log file as being scanned for these open ports. There was no evidence of any peer to

peer file sharing occurring over these ports, and only the incoming 'SYN' packets were observed.

Table 6 shows hosts with destination port entries in the log file of 1214, which is another well known peer to peer port used. There was clear evidence of file sharing activity occurring over this port.

Table - 11

peer to peer hosts – destination port 1214			
999.888.88.162	999.888.91.172	999.888.97.72	999.888.97.141
999.888.150.133	999.888.150.220	999.888.194.13	999.888.194.237
999.888.196.69	999.888.204.102	999.888.205.22	999.888.207.26
999.888. 209.22	999.888.220.54	999.888.221.174	999.888.221.18
999.888.228.6	999.888.229.42	999.888.229.142	999.888.233.114
999.888.233.190	999.888.235.78	999.888.241.90	999.888.241.94
999.888.241.114	999.888.245.214	999.888.249.134	999.888.249.146
999.888.250.174			

In addition to these well known ports being involved in peer to peer activity, the following University hosts given below are being used to share files via Kazaa from customised port numbers of 3584 and 2708. For more information on kazaa is available at: http://www.kazaa.com/us/index.php

In these particular cases the communication captured was between the University host and a single external source host as shown below:

```
03/14-00:25:08.095165 217.159.50.60:1047 -> 999.888.253.82:3584
TCP TTL:113 TOS:0x0 ID:24557 IpLen:20 DgmLen:440 DF
****P*** Seq: 0x39DF7A0A  Ack: 0x0  Win: 0x2000  TcpLen: 20

47 45 54 20 2F 2E 68 61 73 68 3D 64 31 31 65 66    GET /.hash=d11ef
38 65 66 62 33 62 64 38 64 66 62 65 31 37 37 38    8efb3bd8dfbe1778
31 36 36 31 65 61 66 38 39 35 36 36 30 39 65 65    1661eaf8956609ee
39 64 62 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F    9db HTTP/1.1..Ho
73 74 3A 20 31 33 30 2E 38 35 2E 32 35 33 2E 38    st: 999.888.253.8
32 3A 33 35 38 34 0D 0A 55 73 65 72 41 67 65 6E    2:3584..UserAgen
74 3A 20 4B 61 7A 61 61 43 6C 69 65 6E 74 20 4E    t: KazaaClient N
6F 76 20 20 33 20 32 30 30 32 20 32 30 3A 32 39    ov  3 2002 20:29
3A 30 33 0D 0A 58 2D 4B 61 7A 61 61 2D 55 73 65    :03..X-Kazaa-Use
72 6E 61 6D 65 3A 20 62 6F 62 62 79 6F 6E 65 0D    rname: bobbyone.
0A 58 2D 4B 61 7A 61 61 2D 4E 65 74 77 6F 72 6B    .X-Kazaa-Network
3A 20 4B 61 5A 61 41 0D 0A 58 2D 4B 61 7A 61 61    : KaZaA..X-Kazaa
2D 49 50 3A 20 31 39 32 2E 31 36 38 2E 30 2E 37    -IP: 192.168.0.7
38 3A 31 32 31 34 0D 0A 58 2D 4B 61 7A 61 61 2D    8:1214..X-Kazaa-
53 75 70 65 72 6E 6F 64 65 49 50 3A 20 36 36 2E    SupernodeIP: 66.
36 37 2E 38 37 2E 31 35 32 3A 33 35 31 34 0D 0A    67.87.152:3514..
52 61 6E 67 65 3A 20 62 79 74 65 73 3D 33 35 32    Range: bytes=352
37 35 35 38 39 33 2D 33 36 30 37 31 30 31 34 33    755893-360710143
0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 63 6C    ..Connection: cl
6F 73 65 0D 0A 58 2D 4B 61 7A 61 61 2D 58 66 65    ose..X-Kazaa-Xfe
72 49 64 3A 20 31 31 30 38 37 32 34 35 0D 0A 58    rId: 11087245..X
2D 4B 61 7A 61 61 2D 58 66 65 72 55 69 64 3A 20    -Kazaa-XferUid:
37 32 41 42 54 5A 59 56 54 31 54 41 53 75 4C 72    72ABTZYVT1TASuLr
46 31 4A 33 59 52 5A 6A 55 54 44 4E 78 50 76 33    F1J3YRZjUTDNxPv3
6C 5A 64 4E 76 48 37 37 6B 49 73 3D 0D 0A 0D 0A    lZdNvH77kIs=....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

```
03/14-04:52:31.811408 212.150.128.10:41965 -> 999.888.238.22:2708
TCP TTL:108 TOS:0x0 ID:33641 IpLen:20 DgmLen:436 DF
****P*** Seq: 0x302F780A  Ack: 0x0  Win: 0x2000  TcpLen: 20

47 45 54 20 2F 2E 68 61 73 68 3D 64 66 39 64 64     GET /.hash=df9dd
34 34 66 32 37 39 35 64 32 34 31 35 65 36 38 30     44f2795d2415e680
32 39 62 66 36 39 31 66 35 38 64 36 66 31 62 62     29bf691f58d6f1bb
66 64 66 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F     fdf HTTP/1.1..Ho
73 74 3A 20 31 33 30 2E 38 35 2E 32 33 38 2E 32     st: 999.888.238.2
32 3A 32 37 30 38 0D 0A 55 73 65 72 41 67 65 6E     2:2708..UserAgen
74 3A 20 4B 61 7A 61 61 43 6C 69 65 6E 74 20 4E     t: KazaaClient N
6F 76 20 20 33 20 32 30 30 32 20 32 30 3A 32 39     ov  3 2002 20:29
3A 30 33 0D 0A 58 2D 4B 61 7A 61 61 2D 55 73 65     :03..X-Kazaa-Use
72 6E 61 6D 65 3A 20 61 69 74 61 6E 0D 0A 58 2D     rname: aitan..X-
4B 61 7A 61 61 2D 4E 65 74 77 6F 72 6B 3A 20 4B     Kazaa-Network: K
61 5A 61 41 0D 0A 58 2D 4B 61 7A 61 61 2D 49 50     aZaA..X-Kazaa-IP
3A 20 31 30 2E 31 39 34 2E 35 2E 34 30 3A 31 35     : 10.194.5.40:15
34 31 0D 0A 58 2D 4B 61 7A 61 61 2D 53 75 70 65     41..X-Kazaa-Supe
72 6E 6F 64 65 49 50 3A 20 36 36 2E 33 30 2E 32     rnodeIP: 66.30.2
34 38 2E 31 36 3A 31 36 31 34 0D 0A 52 61 6E 67     48.16:1614..Rang
65 3A 20 62 79 74 65 73 3D 35 36 30 31 38 30 31     e: bytes=5601801
39 36 2D 35 36 30 37 35 30 37 33 37 0D 0A 43 6F     96-560750737..Co
6E 6E 65 63 74 69 6F 6E 3A 20 63 6C 6F 73 65 0D     nnection: close.
0A 58 2D 4B 61 7A 61 61 2D 58 66 65 72 49 64 3A     .X-Kazaa-XferId:
20 31 34 31 38 35 36 32 34 0D 0A 58 2D 4B 61 7A     14185624..X-Kaz
61 61 2D 58 66 65 72 55 69 64 3A 20 78 74 38 69     aa-XferUid: xt8i
74 71 78 69 49 6D 34 33 62 73 30 50 66 58 62 2F     tqxiIm43bs0PfXb/
54 65 4A 35 4E 43 4A 72 4E 74 4C 71 72 34 72 58     TeJ5NCJrNtLqr4rX
52 74 36 58 74 36 73 3D 0D 0A 0D 0A                 Rt6Xt6s=....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The following host is also being used as a Kazaa client as above but here communication is observed between six different sources:

```
03/18-19:31:35.608543 148.63.204.75:4075 -> 999.888.247.174:1382
TCP TTL:112 TOS:0x0 ID:53741 IpLen:20 DgmLen:439 DF
****P*** Seq: 0x7EECD00A  Ack: 0x0  Win: 0x2000  TcpLen: 20

47 45 54 20 2F 2E 68 61 73 68 3D 36 32 33 34 32     GET /.hash=62342
35 39 36 65 39 38 62 31 62 64 64 65 62 37 31 38     596e98b1bddeb718
66 62 39 30 64 38 65 37 33 33 30 37 35 31 34 33     fb90d8e733075143
63 30 38 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F     c08 HTTP/1.1..Ho
73 74 3A 20 31 33 30 2E 38 35 2E 32 34 37 2E 31     st: 999.888.247.1
37 34 3A 31 33 38 32 0D 0A 55 73 65 72 41 67 65     74:1382..UserAge
6E 74 3A 20 4B 61 7A 61 61 43 6C 69 65 6E 74 20     nt: KazaaClient
4E 6F 76 20 20 33 20 32 30 30 32 20 32 30 3A 32     Nov  3 2002 20:2
39 3A 30 33 0D 0A 58 2D 4B 61 7A 61 61 2D 55 73     9:03..X-Kazaa-Us
65 72 6E 61 6D 65 3A 20 68 6F 6E 64 61 31 30 31     ername: honda101
0D 0A 58 2D 4B 61 7A 61 61 2D 4E 65 74 77 6F 72     ..X-Kazaa-Networ
6B 3A 20 4B 61 5A 61 41 0D 0A 58 2D 4B 61 7A 61     k: KaZaA..X-Kaza
61 2D 49 50 3A 20 31 34 38 2E 36 33 2E 32 30 34     a-IP: 148.63.204
2E 37 35 3A 31 32 31 34 0D 0A 58 2D 4B 61 7A 61     .75:1214..X-Kaza
61 2D 53 75 70 65 72 6E 6F 64 65 49 50 3A 20 31     a-SupernodeIP: 1
32 2E 32 33 38 2E 35 39 2E 36 35 3A 31 32 31 34     2.238.59.65:1214
0D 0A 52 61 6E 67 65 3A 20 62 79 74 65 73 3D 31     ..Range: bytes=1
```

```
31 31 38 34 38 31 31 2D 31 36 37 37 37 32 31 35     1184811-16777215
0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 63 6C     ..Connection: cl
6F 73 65 0D 0A 58 2D 4B 61 7A 61 61 2D 58 66 65     ose..X-Kazaa-Xfe
72 49 64 3A 20 37 33 37 36 32 38 36 0D 0A 58 2D     rId: 7376286..X-
4B 61 7A 61 61 2D 58 66 65 72 55 69 64 3A 20 50     Kazaa-XferUid: P
58 79 69 70 6A 2B 6E 6C 45 38 77 75 61 42 46 49     Xyipj+nlE8wuaBFI
43 44 61 73 6B 4A 34 75 48 50 30 69 35 73 64 78     CDaskJ4uHP0i5sdx
68 49 41 36 77 35 6D 43 56 6F 3D 0D 0A 0D 0A        hIA6w5mCVo=....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The following University host is being used as a Gnutella client on a custom port of 6011, which is one of the port numbers used by X-Windows as standard. In this case the University host seems to be operating as an Ultrapeer for other Gnutella clients. For detailed information on Gnutella operation and the use of ultrapeers see the following url: http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm

```
03/14-15:03:57.222814 148.64.157.178:2490 -> 999.888.207.2:6011
TCP TTL:112 TOS:0x0 ID:22713 IpLen:20 DgmLen:331 DF
****P*** Seq: 0x2CBBBA0A  Ack: 0x0  Win: 0x2000  TcpLen: 20

47 4E 55 54 45 4C 4C 41 20 43 4F 4E 4E 45 43 54     GNUTELLA CONNECT
2F 30 2E 36 0D 0A 58 2D 55 6C 74 72 61 70 65 65     /0.6..X-Ultrapee
72 3A 20 54 72 75 65 0D 0A 55 73 65 72 2D 41 67     r: True..User-Ag
65 6E 74 3A 20 42 65 61 72 53 68 61 72 65 20 34     ent: BearShare 4
2E 32 2E 34 0D 0A 4D 61 63 68 69 6E 65 3A 20 31     .2.4..Machine: 1
2C 31 33 2C 35 31 31 2C 31 2C 31 31 30 30 0D 0A     ,13,511,1,1100..
50 6F 6E 67 2D 43 61 63 68 69 6E 67 3A 20 30 2E     Pong-Caching: 0.
31 0D 0A 58 2D 51 75 65 72 79 2D 52 6F 75 74 69     1..X-Query-Routi
6E 67 3A 20 30 2E 31 0D 0A 48 6F 70 73 2D 46 6C     ng: 0.1..Hops-Fl
6F 77 3A 20 31 2E 30 0D 0A 4C 69 73 74 65 6E 2D     ow: 1.0..Listen-
49 50 3A 20 31 34 38 2E 36 34 2E 31 35 37 2E 31     IP: 148.64.157.1
37 38 3A 36 33 34 36 0D 0A 52 65 6D 6F 74 65 2D     78:6346..Remote-
49 50 3A 20 31 33 30 2E 38 35 2E 32 30 37 2E 32     IP: 999.888.207.2
0D 0A 47 47 45 50 3A 20 30 2E 35 0D 0A 42 65 61     ..GGEP: 0.5..Bea
72 43 68 61 74 3A 20 31 2E 30 0D 0A 46 50 2D 41     rChat: 1.0..FP-A
75 74 68 2D 43 68 61 6C 6C 65 6E 67 65 3A 20 36     uth-Challenge: 6
4D 4E 46 4C 36 43 57 34 55 43 35 47 42 48 51 58     MNFL6CW4UC5GBHQX
50 41 4D 50 50 5A 4C 5A 59 4A 43 54 4E 46 35 0D     PAMPPZLZYJCTNF5.
0A 0D 0A                                            ...
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Detects

The top ten detects in descending order, from Table 15 (excluding spp_portscan alerts) will be briefly described in this section:

TCP SRC and DST outside network

This signature detects packets containing both source and destination addresses that are not part of the University network address space. Under normal circumstances this should not actually occur.
What appears to have happened to generate this signature is that a University host(s) has adopted spoofed source addresses to connect to various real external targets. The reasoning behind this is not known but given the changing source address network ranges occurring within very short time periods, it appears that a scanning tool such as nmap may have been used.

SMB Name Wildcard
alert udp any any -> any 137 (msg:"SMB Name Wildcard";
content:"CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|0000|";)
This signature detects Microsoft NetBIOS name queries occurring and destined for port 137. In MS Windows environment this is normal activity, however when observed occurring from external addresses towards the internal network, it is probably an enumeration attempt to gather NetBIOS name table information. This can be useful information to someone wishing to gain unauthorised access to your network. This enumeration is what appears to be happening to the University network.

Watchlist 000220 IL-ISDNNET-990517
This signature is designed to detect traffic emanating from a particular network range of addresses. In this particular case the range is 212.179.0 /17 allocated to ISDN Net Ltd. As described at http://ftp.u-picardie.fr/mirror/ftp.nic.fr/documents/ripe-local-ir/oldchargingfiles/allocs-Nov1998
The signature does not false positive (indirectly perhaps, when this range of addresses is spoofed) and much of this traffic has been detected.

Tiny Fragments - Possible Hostile Activity
alert tcp any any -> any any (minfrag: 256; msg: "Tiny fragments detected, possible hostile activity";)
This is not an actual signature but a pre-processor available with Snort which also analyses the network traffic. In this case the pre-processor is called Minifrag.
A threshold can be set for a fragmented packet which when matched will trigger the alert. Some attackers fragment their packets to avoid detection with such tools as Fragroute etc. It is generally assumed that there is almost no commercial network equipment which will generate fragments smaller than 256-bytes. This is the best setting to configure minifrag to work effectively.
On the University network two hosts were responsible for all of the 555 detects and it is recommended that the University investigate the source of this traffic.
999.888.194.125:0 > 209.126.191.143:0

High port 65535 tcp - possible Red Worm – traffic
This signature is designed to trigger on TCP packets set to source or destination port 65535 and is in response to the Adore or red worm.
Adore/Red attacks vulnerabilities in rpc.statd, bind, LPRng, and wuftpd26.
The worm compiles a trojan'd klogd and this is then set running on port 65535 waiting for an incoming packet with a data size of 77 bytes.
There may be some infection within the campus network and this cannot be ruled out on the log file data alone, though some false positives have been identified in the 'Alert file analysis' section.

CS WEBSERVER - external web traffic
This must be a custom signature designed to alert to traffic targeted at the CS Webserver on port 80. It is not known what the CS refers to, but the only University address given in the alerts is 999.888.100.165.

TFTP - Internal TCP connection to external tftp server
This signature alerts to a University host using TCP to connect to an external TFTP server on target port 69. The signature also catches both sides of the connection, so we also see the external TFTP server port 69 replying to the university host.

SUNRPC highport access!
alert tcp any any -> $HOME_NET 32771 (msg: "Attempted Sun RPC high port access";)
alert udp any any -> $HOME_NET 32771 (msg: "Attempted Sun RPC high port access";)
This signature could be either TCP or UDP and is not specified in the alert log file data. This signature triggers on a connection to a university host port 32771. This port is associated with the Sun RPC portmapper service. Normally, the rpcbind service only listens on port 111. Under Solaris, the rpcbind service also listens under port 32771, which sometimes allows attackers to bypass packet filtering.
This signature(s) seems to have triggered many false positives as discussed in the 'alert file analysis' section.

999.888.30.4 activity
This is a custom signature designed to alert to traffic targeted at the University host 999.888.30.4. There doesn't seem to be any further criteria set in the signature as traffic to many ports is observed through this single signature message.
The host has been identified as a possible Novell Netware system running an Apache Web Server. This deduction was made because the traffic was observed targeting port 51443, which is the default port Apache configures HTTPS to use when also running Netware Enterprise Server. Also another Netware server was previously identified as host 999.888.30.3.

Null scan!
alert tcp any any -> $HOME_NET any (msg:"NULL Scan"; flags: 0;)
This signature alerts to TCP packets targeted at the University network with no flags set in the packets at all. This is not standard TCP/IP behaviour and as such has a very low false positive rate. Thus it is almost certainly being used as an information gathering technique by external hosts. These should be filtered at the gateway if possible to eliminate there effectiveness.

Top Talkers

- Scans

There were 177569 total entries in the scans_file.

The following table shows the top ten scan signatures found in the concatenated scans_file. The sources are the number of different source addresses generating the scan signature, and the dests are the number of different destination addresses receiving the scan signature.

Table - 12

Position	Signature	Hits	Sources	Dests
1	spp_portscan: UDP scan	108874	243	83933
2	spp_portscan: TCP ******S* scan	52560	286	32494
3	spp_portscan: TCP ******** scan	6274	97	541
4	spp_portscan: TCP *******F scan	2680	1295	690
5	spp_portscan: TCP 12****S* scan	931	244	109
6	spp_portscan: TCP **U**RS* scan	778	18	217
7	spp_portscan: TCP **U**R*F scan	671	11	186
8	spp_portscan: TCP **U**RSF scan	363	10	151
9	spp_portscan: TCP **U**R** scan	360	8	130
10	spp_portscan: TCP **U*P*S* scan	277	9	116

The following tables show the top ten source and destination addresses found within the concatenated scans_file log file.

Table - 13

Position	Source IP	Hits
1	999.888.70.176	9910
2	999.888.239.202	9674
3	999.888.196.179	8471
4	999.888.88.134	7615
5	999.888.1.3	7417
6	999.888.88.180	5373
7	80.14.116.185	4849
8	80.14.116.162	4439
9	999.888.168.82	3407
10	999.888.97.21	3343

Table - 14

Position	Destination IP	Hits
1	999.888.234.54	1188
2	24.159.70.120	682
3	62.79.69.54	491
4	192.26.92.30	348
5	999.888.249.194	309
6	999.888.195.67	277
7	64.217.142.50	275
8	12.221.37.190	272
9	999.888.202.214	256
10	205.231.29.244	206

- Alerts

There were 477713 total entries in the alert_file The following table shows all of the alert signatures found within the concatenated alert_file log file in descending order of total alert count.

Table - 15

Position	Alert Signature	Count
1	TCP SRC and DST outside network	195486
2	SMB Name Wildcard	104321
3	Watchlist 000220 IL-ISDNNET-990517	38527
4	spp_portscan: portscan status from	32108
5	Tiny Fragments - Possible Hostile Activity	28534
6	spp_portscan: PORTSCAN DETECTED from	12425
7	spp_http_decode: IIS Unicode attack detected	12374
8	spp_portscan: End of portscan from	12303
9	High port 65535 tcp - possible Red Worm - traffic	9015
10	CS WEBSERVER - external web traffic	8109
11	TFTP - Internal TCP connection to external tftp server	4183
12	spp_http_decode: CGI Null Byte attack detected	2947
13	SUNRPC highport access!	2748
14	999.888.30.4 activity	2331
15	Null scan!	2199
16	High port 65535 udp - possible Red Worm - traffic	1865
17	Incomplete Packet Fragments Discarded	1737
18	IDS552/web-iis_IIS ISAPI Overflow ida nosize	932
19	Queso fingerprint	932
20	Watchlist 000222 NET-NCFC	816
21	Port 55850 tcp - Possible myserver activity - ref. 010313-1	769
22	External RPC call	757
23	999.888.30.3 activity	409
24	CS WEBSERVER - external ftp traffic	312
25	FTP DoS ftpd globbing	224
26	SNMP public access	214
27	EXPLOIT x86 NOOP	197
28	Possible trojan server activity	195
29	IRC evil - running XDCC	118
30	NMAP TCP ping!	111
31	IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	110
32	EXPLOIT x86 setuid 0	93

33	NIMDA - Attempt to execute cmd from campus host	78
34	EXPLOIT x86 setgid 0	44
35	EXPLOIT x86 stealth noop	41
36	Back Orifice	32
37	DDOS shaft client to handler	22
38	TFTP - Internal UDP connection to external tftp server	20
39	TFTP - External TCP connection to internal tftp server	15
40	Notify Brian B. 3.56 tcp	13
41	Probable NMAP fingerprint attempt	13
42	Notify Brian B. 3.54 tcp	10
43	SMB C access	10
44	FTP passwd attempt	9
45	Port 55850 udp - Possible myserver activity - ref. 010313-1	3
46	EXPLOIT NTPDX buffer overflow	2

The following tables show the top ten source and destination addresses found within the concatenated alert_file log file.
N.B. The ?.?.?.? destination IP address is the unknown target of the spp_portscan activity found within the alert log files.

Table - 16

Position	Source IP	Hits
1	999.888.239.202	43482
2	212.179.99.189	7991
3	212.179.33.82	5178
4	212.179.98.134	4850
5	212.179.16.225	4524
6	80.60.247.181	4243
7	212.179.126.3	3549
8	12.38.10.4	3508
9	999.888.88.193	2092
10	999.888.70.176	1944

Table - 17

Position	Destination IP	Hits
1	208.253.114.222	105321
2	?.?.?.?	56839
3	131.118.254.39	33095
4	208.225.90.120	29997
5	65.116.88.75	18530
6	999.888.100.165	8575
7	64.251.196.146	8054
8	999.888.108.45	7992
9	999.888.210.238	5494
10	999.888.210.234	4362

-Out of Spec

There were 7154 total entries in the oos_file. The following tables (8 and 9) show the top ten source and destination addresses found within the concatenated oos_file log file.

Table - 18

Position	Source IP	Hits
1	68.54.93.181	1057
2	66.140.25.157	332
3	209.191.132.40	281
4	216.95.201.41	147
5	216.95.201.40	142
6	212.186.78.246	129
7	194.109.249.60	118
8	216.95.201.25	101
9	217.159.50.60	92
10	12.218.222.121	84

Table - 19

Position	Destination IP	Hits
1	999.888.6.7	1093
2	999.888.24.21	355
3	999.888.6.40	354
4	999.888.24.23	334
5	999.888.6.47	313
6	999.888.207.2	284
7	999.888.201.22	281
8	999.888.24.44	275
9	999.888.24.22	257
10	999.888.205.222	183

The following tables (15 and 16) show the top ten source and destination ports found within the concatenated oos_file log file.

Table - 20

Position	Source Port	Hits
1	20	56
2	993	26
3	2828	19
4	143	17
5	1029	14
6	1434	13
7	1522	13
8	1025	12
9	0	10
10	3111	10

Table – 21

Position	Destination Port	Hits
1	25	1696
2	110	1058
3	6346	965
4	80	641
5	4662	504
6	6011	283
7	1214	152
8	2419	128
9	3584	92
10	1470	68

Source Address Information

Lookup #1 - 212.179.126.3

This source was chosen because one of the custom signatures found in the alert_file log file was [**] Watchlist 000220 IL-ISDNNET-990517 [**], and one of the most active source addresses within this network range was 212.179.126.3. The allocation of 990517 to IL-ISDN-NET can be found detailed at the following url:

http://ftp.u-picardie.fr/mirror/ftp.nic.fr/documents/ripe-local-ir/oldchargingfiles/allocs-Nov1998

Online lookup performed by http://centralops.net/co/

Address lookup
canonical name **bzq-179-126-3.cust.bezeqint.net**

Domain Whois record

Querying whois.internic.net with "dom bezeqint.net"...

Whois Server Version 1.3
Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

Domain Name: BEZEQINT.NET
Registrar: NETWORK SOLUTIONS, INC.
Whois Server: whois.networksolutions.com
Referral URL: http://www.networksolutions.com
Name Server: NS1.BEZEQINT.NET
Name Server: NS2.BEZEQINT.NET
Status: ACTIVE
Updated Date: 05-nov-2001
Creation Date: 04-nov-1998
Expiration Date: 03-nov-2010
Registrant:
Bezeq International (BEZEQINT2-DOM)
40 Hashacham St.
Petach Tikva

Submitted by: Antony Gummery Date Submitted: 05/04/2003

Israel,49170
IL
Domain Name: BEZEQINT.NET
Administrative Contact:
yuval, keinan (STQDHBUQSI) hostmaster@BEZEQINT.NET
bezeq international
40 Hashacham St.
IL
972-3-9203010 972-3-9203033
Technical Contact:
Peer, Tomer (TP5909) hostmaster@BEZEQINT.NET
ISDN Net-Bezeqint
Hashacham 40
IL
972-3-9257778 972-3-9220135
Billing Contact:
Bezeq International (BI3752-ORG) billingdomains@BEZEQINT.CO.IL
Bezeq International
40 hashacham Street
Petach-Tikva
IL
972-1-800-800-110 fax: 972-3-9257369
Record last updated on 18-Sep-2002.
Record expires on 04-Nov-2010.
Record created on 04-Nov-1998.
Database last updated on 3-Apr-2003 18:51:17 EST.
Domain servers in listed order:
NS1.BEZEQINT.NET 192.115.106.10
NS2.BEZEQINT.NET 192.115.106.11

Network Whois record

Querying whois.arin.net with "**212.179.126.3**"...

Querying whois.ripe.net with "**212.179.126.3**"...

% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum: **212.179.126.0 - 212.179.126.127**
netname: **BARAM-LTD**
mnt-by: INET-MGR
descr: BARAM-NET
country: IL
admin-c: YO141-RIPE
tech-c: YO141-RIPE
status: ASSIGNED PA
notify: hostmaster@isdn.net.il
changed: hostmaster@isdn.net.il 20010715
source: RIPE
route: 212.179.64.0/18
descr: ISDN Net Ltd.
origin: AS8551
notify: hostmaster@bezeqint.net
mnt-by: AS8551-MNT
changed: hostmaster@bezeqint.net 20020618

Submitted by: Antony Gummery Date Submitted: 05/04/2003

source: RIPE
person: Yair Ovadia
address: Bezeq Inernational
address: hashacham 40
address: Petach Tiqua
address: Israel
phone: +972-3-9203010
phone: +972-3-9203005
e-mail: hostmaster@bezeqint.net
nic-hdl: YO141-RIPE
changed: hostmaster@bezeqint.net 20010913 source: RIPE

Lookup #2 – 216.95.201.41

This source was chosen because it was one of the top source addresses found in the oos_file log file, and was identified as a known UCE spammer. From the logs this source appeared to be attempting to connect to University mail servers. Confirmation of this source and others from the same domain can be found at: http://spamcop.net/w3m?action=checkblock&ip=216.95.201.41

Online lookup performed by http://centralops.net/co/

Address lookup

canonical name **smtp31.jsuati.com**

Domain Whois record

Querying whois.internic.net with "**dom jsuati.com**"...

Whois Server Version 1.3

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

Domain Name: JSUATI.COM
Registrar: TUCOWS, INC.
Whois Server: whois.opensrs.net
Referral URL: http://www.opensrs.org
Name Server: NS1.JSUATI.COM
Name Server: NS2.JSUATI.COM
Status: REGISTRAR-LOCK
Updated Date: 29-jan-2003
Creation Date: 27-jan-2003
Expiration Date: 27-jan-2004
Registrant:
1505820 Ontario Inc
610 Ford Drive Unit #1
Oakville, ON L6J 7V2
CA
Domain name: **JSUATI.COM**
Administrative Contact:
Administrator, Network admin@jsuati.com
610 Ford Drive Unit #1
Oakville, ON L6J 7V2

Submitted by: Antony Gummery Date Submitted: 05/04/2003

CA
(905) 337 8616 Fax: (905) 337 2882
Technical Contact:
Administrator, Network admin@jsuati.com
610 Ford Drive Unit #1
Oakville, ON L6J 7V2
CA
(905) 337 8616 Fax: (905) 337 2882
Registrar of Record: TUCOWS, INC.
Record last updated on 29-Jan-2003.
Record expires on 27-Jan-2004.
Record Created on 27-Jan-2003.

Domain servers in listed order:
NS1.JSUATI.COM 216.95.201.5
NS2.JSUATI.COM 216.95.201.6

Network Whois record

Querying whois.arin.net with "**216.95.201.41**"...

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@wcom.com
OrgNOCHandle: OA12-ARIN
OrgNOCName: UUnet Technologies, Inc., Technologies
OrgNOCPhone: +1-800-900-0241
OrgNOCEmail: help4u@wcom.com
OrgTechHandle: SWIPP-ARIN
OrgTechName: swipper
OrgTechPhone: +1-800-900-0241
OrgTechEmail: swipper@uu.net

Submitted by: Antony Gummery Date Submitted: 05/04/2003

Lookup #3 – 80.60.247.181

This host was chosen as it was number 6 in the top ten listing for alert_file source addresses. This host was observed generating many spp_portscan and other nmap and queso fingerprint alerts, and seemed particularly interested in the University host 999.888.234.54 as a target for this activity.

Online lookup performed by http://centralops.net/co/

Address lookup

canonical name **ip503cf7b5.speed.planet.nl**

Domain Whois record

nl = Netherlands

Querying whois.nic.nl with "**planet.nl**"...

Rights restricted by copyright. See http://www.domain-registry.nl/whois.php
Domain name: **planet.nl** (first domain)
Status: active
Registrant:
Planet Media Group N.V.
Printerweg 14 32
3821 AD AMERSFOORT
Netherlands
Domicile:
N/A
Committed to ADR: yes
Administrative contact:
R. Niamat
+31 33 4540400
postmaster@planet.nl
Registrar:
Planet Media Group N.V
Printerweg 14 -32
3821 AD AMERSFOORT
Netherlands
Technical contact:
. Planet Internet Domain Beheer
+31 33 4540400
domain-ops@planet.nl
Technical contact:
. Domein Beheer Planet Internet
+31 33 4540400
domein-ops@planet.nl
Domain nameservers:
ns08.wxs.nl 195.121.1.40
ns09.wxs.nl 195.121.1.67
ns5.wxs.nl 192.215.32.19

Network Whois record

Querying whois.arin.net with "80.60.247.181"...

Querying whois.ripe.net with "80.60.247.181"...

Submitted by: Antony Gummery Date Submitted: 05/04/2003

```
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      80.60.0.0 - 80.60.255.255
netname:      NL-PMG-ADSL
descr:        ADSL5
country:      NL
admin-c:      MRAA-RIPE
tech-c:       PT978-RIPE
status:       ASSIGNED PA
mnt-by:       AS8737-MNT
changed:      lir@planet.nl 20030402
source:       RIPE

route:        80.60.0.0/15
descr:        PIADDR
origin:       AS8737
mnt-by:       AS8737-MNT
changed:      lir@planet.nl 20010618
source:       RIPE
role:         Planet Technologies
address:      Stationsstraat 115 (visit address)
address:      P.O. box 1042
address:      3800 BA Amersfoort
address:      The Netherlands
phone:        +31 33 45 40 550
e-mail:       lir@planet.nl
nic-hdl:      PT978-RIPE
admin-c:      MRAA-RIPE
tech-c:       NKOL-RIPE
remarks:      Please mail security issues to: security@planet.nl
remarks:      Please mail abuse issues to: abuse@planet.nl
notify:       lir@planet.nl
mnt-by:       AS8737-MNT
changed:      lir@planet.nl 20030402
source:       RIPE
person:       Marc Raaff
address:      Planet Technologies
address:      Stationsstraat 115 (visit address)
address:      P.O. box 1042
address:      3800 BA Amersfoort
address:      The Netherlands
phone:        +31 33 45 40 550
e-mail:       lir@planet.nl
nic-hdl:      MRAA-RIPE
remarks:      Please mail security issues to: security@planet.nl
remarks:      Please mail abuse issues to: abuse@planet.nl
notify:       lir@planet.nl
mnt-by:       AS8737-MNT
changed:      lir@planet.nl 20030327
source:       RIPE
```

Lookup #4 – 65.29.135.228

This host was chosen because it was detected attempting to connect to numerous University systems on port 31337, which is the default port of the Back Orifice trojan / remote control program. As such the IDS signature for this attack was triggered. When using the online scan tools from http://centralops.net/co/ some very interesting information was gathered about the services running on this system, as seen below:

Service scan

FTP – 21	220 ProFTPD 1.2.5 Server (ProFTPD) [ratsting.com]
SMTP – 25	220 ratsting.com ESMTP Postfix (1.1.11) (Mandrake Linux)
HTTP – 80	HTTP/1.1 200 OK Date: Fri, 04 Apr 2003 02:37:27 GMT Server: Apache-AdvancedExtranetServer/1.3.26 (Mandrake Linux/6.1mdk) mod_ssl/2.8.10 OpenSSL/0.9.6g PHP/4.2.3 Last-Modified: Mon, 10 Mar 2003 05:26:54 GMT ETag: "23b32-10a-3e6c221e" Accept-Ranges: bytes Content-Length: 266 Connection: close Content-Type: text/html
POP3 – 110	Error: Connection refused
NNTP – 119	Error: Connection refused

Address lookup

canonical name **cpe-65-29-135-228.wi.rr.com**

Domain Whois record

Querying whois.internic.net with "**dom rr.com**"...

Whois Server Version 1.3

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

Domain Name: RR.COM
Registrar: NETWORK SOLUTIONS, INC.
Whois Server: whois.networksolutions.com
Referral URL: http://www.networksolutions.com
Name Server: DNS1.RR.COM
Name Server: DNS2.RR.COM
Name Server: DNS3.RR.COM
Name Server: DNS4.RR.COM
Status: ACTIVE
Updated Date: 24-oct-2002
Creation Date: 01-oct-1996
Expiration Date: 30-sep-2010
Registrant:
Road Runner HoldCo, LLC (RR6-DOM)
13241 Woodland Park Rd
Herndon, VA 20171
US
Domain Name: RR.COM
Administrative Contact, Technical Contact:

Submitted by: Antony Gummery Date Submitted: 05/04/2003

Road Runner (XGUKSSRMIO) abuse@RR.COM
Road Runner
13241 Woodland Park Rd
Herndon, VA 20171
US
703-345-3416 fax: 703-345-2518
Billing Contact:
idNames, Accounting (IA90-ORG) accounting@IDNAMES.COM
idNames from Network Solutions, Inc
440 Benmar
Suite #3325
Houston, TX 77060
US
281-447-1044
Fax- 281-447-1160
Record last updated on 06-Feb-2003.
Record expires on 02-Oct-2010.
Record created on 01-Oct-1996.
Database last updated on 3-Apr-2003 21:26:11 EST.
Domain servers in listed order:

DNS1.RR.COM	**24.30.200.3**
DNS2.RR.COM	**24.30.201.3**
DNS3.RR.COM	**24.30.199.7**
DNS4.RR.COM	**65.24.0.172**

Network Whois record

Querying whois.arin.net with "**65.29.135.228**"...

OrgName: **Road Runner**
OrgID: RRMA
Address: 13241 Woodland Park Road
City: Herndon
StateProv: VA
PostalCode: 20171
Country: US
NetRange: **65.28.0.0 - 65.31.255.255**
CIDR: 65.28.0.0/14
NetName: RR-CENTRAL-2BLK
NetHandle: NET-65-28-0-0-1
Parent: NET-65-0-0-0-0
NetType: Direct Allocation
NameServer: DNS1.RR.COM
NameServer: DNS2.RR.COM
NameServer: DNS3.RR.COM
NameServer: DNS4.RR.COM
Comment: ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate: 2001-02-08
Updated: 2002-08-14
TechHandle: ZS30-ARIN
TechName: ServiceCo LLC
TechPhone: +1-703-345-3416
TechEmail: abuse@rr.com
OrgTechHandle: IPTEC-ARIN
OrgTechName: IP Tech
OrgTechPhone: +1-703-345-3416
OrgTechEmail: abuse@rr.com
OrgAbuseHandle: ABUSE10-ARIN
OrgAbuseName: Abuse

Submitted by: Antony Gummery Date Submitted: 05/04/2003

OrgAbusePhone: +1-703-345-3416
OrgAbuseEmail: abuse@rr.com

Lookup #5 – 208.196.247.133

This host was chosen from the alert_file and was responsible for generating [**] Possible trojan server activity [**] events as result of port 27374 (Subseven Trojan) activity. Indeed the logs appear to show that at least three University hosts were compromised by Subseven and 'controlled' by this external host.

Online lookup performed by http://centralops.net/co/

Address lookup

lookup failed 208.196.247.133
No data

Domain Whois record

Don't have a domain name for which to get a record

Network Whois record

Querying whois.arin.net with "**208.196.247.133**"...

```
UUNET Technologies, Inc. UUNET1996B (NET-208-192-0-0-1)
                                  208.192.0.0 - 208.255.255.255
Cable Bahamas Ltd. UU-208-196-247-D1 (NET-208-196-247-0-1)
                                  208.196.247.0 - 208.196.247.255

# ARIN WHOIS database, last updated 2003-04-03 20:00
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

Querying whois.arin.net with **"!NET-208-196-247-0-1**"...

```
OrgName:    Cable Bahamas Ltd.
OrgID:      CBL-3
Address:    Robinson at Marathon Road, PO. Box CB-13050
Address:    Nassau, BHS
City:
StateProv:
PostalCode:
Country:    BS
NetRange:   208.196.247.0 - 208.196.247.255
CIDR:       208.196.247.0/24
NetName:    UU-208-196-247-D1
NetHandle:  NET-208-196-247-0-1
Parent:     NET-208-192-0-0-1
NetType:    Reassigned
Comment:
RegDate:    2000-03-28
Updated:    2000-03-28
TechHandle: AF67-ARIN
TechName:   Foster, Andre
TechPhone:  242-356-8976
```

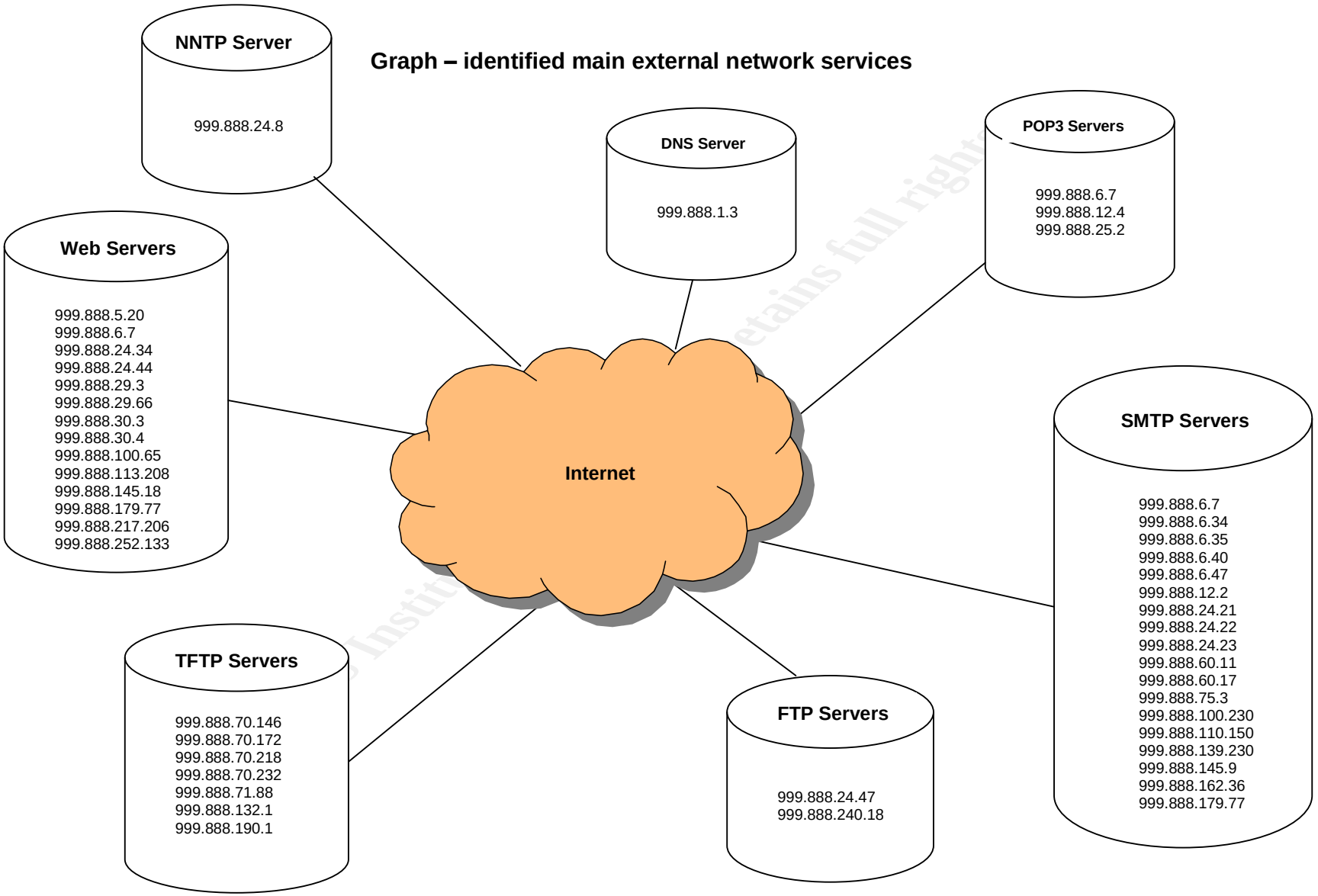
Submitted by: Antony Gummery Date Submitted: 05/04/2003

```
TechEmail:  hostmaster@cablebahamas.com
# ARIN WHOIS database, last updated 2003-04-03 20:00
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

DNS records

196.208.in-addr.arpa	IN	SOA	server:	auth03.ns.uu.net
			email:	hostmaster@uu.net
			serial:	20001039
			refresh:	21600
			retry:	3600
			expire:	1728000
			minimum ttl:	21600

Graph – identified main external network services



Submitted by: Antony Gummery

Date Submitted: 05/04/2003

Anomalous Activity

In the course of analysis of the University log files provided, there are some network services and systems that show signs of possible compromise and/or have been involved in anomalous activity. Where this is suspected, this section will draw attention to these hosts or services, and Gummery Information Security Services recommend that the University investigate these conclusions as soon as possible.

Nimda compromised hosts
The following University hosts are suspected as being infected with the Nimda virus/worm and appear to be actively trying to infect external targets. The signature which alerted to this possibility was: **NIMDA - Attempt to execute cmd from campus host**.
999.888.195.157; 999.888.97.222; 999.888.97.72; 999.888.97.43

SubSeven Trojan compromised hosts
The following University hosts show signs of being compromised by the Subseven Trojan program. The external host given below scanned a small number of University target addresses for target port 27374. Though not initially scanned, the three University hosts were observed as source addresses communicating with the external host from a source port of 27374.
999.888.137.1:27374 > 208.196.247.133
999.888.137.17:27374 > 208.196.247.133
999.888.137.33:27374 > 208.196.247.133

Myserver Trojan activity
Some strange traffic was observed between two distinct hosts that requires further investigation to determine if the activity is due to the myserver DDOS Trojan. It would be more worrying if the University host ports were 55850 as this is the default port this program binds to, however the activity should be investigated.
207.254.193.117:55850 > 999.888.250.226:3190
999.888.250.226:3190 > 207.254.193.117:55850

High port 65535 tcp - possible Red Worm – traffic
High port 65535 udp - possible Red Worm – traffic
There may be some systems which have been compromised by the Adore or red worm as described by the signature message. This cannot be conclusively determined by the log file data alone and further investigation should be entered into. See 'Detects' section.

University host 999.888.239.202
The three destination hosts from table 9 of 24.159.70.120; 62.79.69.54 and 12.221.37.190 are all recipients of packets with some strange flag combinations and NULL scans. This host also generated many **[**]Tiny Fragments - Possible Hostile Activity[**]** alerts. The precise nature of the traffic cannot be ascertained from the log data alone, and it is recommended that this host should be investigated more closely

University host 999.888.194.125
The traffic between two specific hosts caused 555 of the following alerts to be generated **[**]Incomplete Packet Fragments Discarded[**]**. This is not obvious malicious activity but can be termed anomalous, and as such further investigation is recommended.
999.888.194.125:0 > 209.126.191.143:0

Peer to Peer activity

Much traffic has been observed from the log files supplied which suggests that peer to peer file sharing programs and associated activity is rife on the campus network. Though these programs can be made secure with careful configuration, (beyond the scope of this report), they do pose a potential threat to the security of the University network. This is really a matter for the University Security Policy to address and no recommendations for this policy are being made here, simply information into the occurrence and scope of this activity. Active programs and the corresponding ports being used in this respect are:

Kazaa (ports 1214*; 3584****; 2708***; 1382***)**
Gnutella (6011*; 65535)**
GNUnet (src-6257*; tgt-65535*)
WinMX (6699*)

University host 999.888.196.179

The second top source host in table 8 is 999.888.196.179. This host exhibits similar UDP traffic to that of the aforementioned top source, but with a different source/destination port of 22321. The exact nature of the use of this port is unclear and no conclusive information has been forthcoming on this, except to say that it is very likely peer to peer traffic that is being observed. This traffic was also observed for three other University source hosts **999.888.88.134; 999.888.88.180 and 999.888.168.82.**

IRC activity

As with the peer to peer activity observed above, the use of Internet Relay Chat (IRC) is a matter for the overall University Security policy to address. These programs and services have been known to be exploited, and systems compromised through their use. Table 7, 8 and the preceding paragraph details these findings and break them down into ports, University hosts and external IRC servers actively being used.

Defensive Recommendations

Gummery Information Security Services has not been given access to the University gateway router or firewall configuration, and so any defensive recommendations or considerations given in this section may already have been considered or implemented by the University. The recommendations given here are based on the information detailed in the 'Relational Analysis', 'Detects' and 'Anomalous Activity' sections.

The University SMTP servers should be secured from being used as spam relays given the findings detailed in the 'Relational Analysis' section regarding jsuati.com and other known spam agents. A selection of software and tips for helping prevent spam is listed here:
http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2880727,00.html
http://www.bagley.org/~doug/spam/dirty.shtml#relays

The SMTP servers should also be well maintained and patched if they are running Sendmail, as recent vulnerabilities on this platform have been disclosed. The latest of which can be found here: http://www.cert.org/advisories/CA-2003-12.html

Much general scanning and alert activity was seen directed at the Microsoft Windows default ports of 135, 137 (such as [**] SMB Name Wildcard [**] Table 10), 139 and 445 as is encountered by most internet connected systems. However this also means that these ports must be filtered by the gateway to ensure internal system integrity, if the Microsoft Windows platform is deployed at the University. Port 445 used by Windows 2000 systems and later is often overlooked when configuring the gateway to block ports.

Block access to the rpc portmapper service. This service often known as the Sun RPC portmapper. Normally, the this service only listens on port 111. Under Solaris, the rpcbind service also listens under port 32771, which sometimes allows attackers to bypass packet filtering. There is plenty of evidence of both port 111 and 32771 being actively probed and so these should be blocked at the gateway.

SNMP is another actively probed service from external sources to port 161. If SNMP is in operation on the campus network (there is no evidence to suggest that it is), then as well as blocking port 161 at the gateway, non standard community strings and strong passwords should be employed.

Information on configuring port blocking on cisco products can be found here:
http://www.sans.org/rr/firewall/blocking_cisco.php

There is also evidence of TFTP Servers being active on the campus network. This is an inherently insecure method of data transfer and introduces security considerations for the configuration of the system hosting the application. A CERT advisory and some common problems encountered with TFTP are discussed at the following urls:
http://www.cert.org/advisories/CA-1991-18.html
http://216.239.57.100/search?q=cache:AxqlZXP-1uIC:www.netcessity.com/downloads/tftp.pdf+tftp+%2Bsecurity&hl=en&ie=UTF-8

Evidence of the use of the AOL Instant Messenger (AIM) program was observed. As with the peer to peer and IRC issues discussed above, this is matter for the University Security Policy to decide upon. If it is to be allowed, then as with the aforementioned services, a strict usage and configuration policy is recommended.
http://www.securiteam.com/securityreviews/3K5Q1SANFE.html

It is a possibility observed from a small number of log file entries, that POP3 services may be accessible to external systems. One big problem with POP3 is that account information is transferred in plain text. If POP3 must be accessible from the internet, then POP3 over SSL would be the preferred option.
http://security.fi.infn.it/tools/stunnel/index-en.html

A small number of log file entries relating to port 23 (telnet) were observed. No conclusive evidence of Telnet services running on University hosts was found, however GISS recommend the use of SSH if remote login is required.

Analysis Process

The alert and out of spec log files contained munged addresses in the form of MY.NET.x.x instead of the real addresses of the University systems. However the scans log files seemed to contain the actual range of class B addresses of the University which had somehow escaped the 'munging' process. To maintain the confidentiality of the University's address space, and for the purposes of analysis all of the University addresses were changed to 999.888.x.x.

Even without this information on the actual addresses being available in the scan log file headers it would have been possible to work out the actual address ranges by observing one of the packet dumps found within the oos_file:

```
03/18-22:31:44.163533 204.210.11.165:2068 -> 999.888.179.77:80
TCP TTL:111 TOS:0x0 ID:16693 IpLen:20 DgmLen:307 DF
12*APR*F Seq: 0x5DDCDF9  Ack: 0x76F2  Win: 0x5018  TcpLen: 4

00 00 47 45 54 20 2F 6D 61 69 6E 2F 61 77 61 72     ..GET /main/awar
64 73 2F 67 61 6C 61 78 79 2E 67 69 66 20 48 54     ds/galaxy.gif HT
54 50 2F 31 2E 31 0D 0A 41 63 63 65 70 74 3A 20     TP/1.1..Accept:
2A 2F 2A 0D 0A 52 65 66 65 72 65 72 3A 20 68 74     */*..Referer: ht
74 70 3A 2F 2F 64 69 6E 6F 73 61 75 72 69 63 6F     tp://dinosaurico
6E 2E 63 6F 6D 2F 6D 61 69 6E 2F 69 6E 64 65 78     n.com/main/index
2E 68 74 6D 6C 0D 0A 41 63 63 65 70 74 2D 4C 61     .html..Accept-La
6E 67 75 61 67 65 3A 20 65 6E 2D 75 73 0D 0A 41     nguage: en-us..A
63 63 65 70 74 2D 45 6E 63 6F 64 69 6E 67 3A 20     ccept-Encoding:
67 7A 69 70 2C 20 64 65 66 6C 61 74 65 0D 0A 55     gzip, deflate..U
73 65 72 2D 41 67 65 6E 74 3A 20 4D 6F 7A 69 6C     ser-Agent: Mozil
6C 61 2F 34 2E 30 20 28 63 6F 6D 70 61 74 69 62     a/4.0 (compatib
6C 65 3B 20 4D 53 49 45 20 35 2E 35 3B 20 57 69     e; MSIE 5.5; Wi
6E 64 6F 77 73 20 39 38 29 0D 0A 48 6F 73 74 3A     ndows 98)..Host:
20 64 69 6E 6F 73 61 75 72 69 63 6F 6E 2E 63 6F     dinosauricon.co
6D 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 4B     m..Connection: K
65 65 70 2D 41 6C 69 76 65 0D 0A                    ep-Alive..
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

By going to the referrer given in the ascii code it doesn't take too much effort to think of tracing www.dinosauricon.com back to the x.x.179.77 address and get the network range it is part of.

Each of the five separate files for each type was firstly concatenated into one file for each type with a simple command:

```
# cat alert.* > alert_file
# cat scans.* > scans_file
# cat oos_* > oos_file
```

The following perl command was issued to perform the task of changing the University addresses:

```
# perl –pli –e 's/MY\.NET/999.888/g' filename
```

With this action performed each type of concatenated log file could then be analysed. My intention from the start was to be able to analyse the data from a relational

perspective if possible, and so I began searching for the tools with which I could achieve this.

The systems used to run the tools discussed below included a Pentium 4 2.0GHz/512MB RAM /Windows XP computer, which also was used to run a VMWare[1] Mandrake Linux 8.2 server with a MySQL[2] database. A separate Red Hat Linux 7.2 system was also used to help with the processing of the log file data.

Snortsnarf[3] was used to convert the concatenated log files into html output for further analysis. However it was found that this process was very resource intensive and in the case of the concatenated scans_file (11MB), this took 32 hours to complete running on a VMWare client Mandrake Linux 8.2 server assigned a 10GB disk volume and 256MB RAM. The host system was a P4 2.0 GHz, 512MB RAM specified computer. The following command was run:

```
# snortsnarf.pl –rulesfile /etc/snort.conf –d /var/www/html/ /tmp/scansfile
```

Attempting to use this process for the much larger alert_file (55MB), resulted in Snortsnarf reporting 'unexpected alert output' errors and the operation did not complete. Thus a different method was required to analyse the alert file. After some further searching I came across the previously submitted GCIA practical assignment by Brandon Newport[4], which included some custom scripts for parsing the alert type log files and loading the subsequent data into a MySQL database.

The alert_file parsing scripts and the MySQL database ran from a VMWare client Mandrake Linux 8.2 server assigned a 10GB disk volume and 256MB RAM. The host system was a P4 2.0 GHz, 512MB RAM specified computer, running Windows XP Professional.

Rather than simply query the database for statistical data such as total number of alerts, different sources, destinations etc, I decided to use a front-end to give a full view of all the event data in the database in order to build up a mental picture of the activity occurring on the target network. For this I used Microsoft Access to import the table data from the MySQL database, which meant that I had a local .mdb file available for manipulation, improving the performance of this analysis method over maintaining a remote connection to the MySQL database via the MS Access front-end.

The remaining log file type of Out of Spec data needed yet another method applied to perform effective analysis. I attempted to use some popular Snort log analysis tools such as Snortsnarf for this, as well as Snort_sort[5] and WinSnort2html[6], however the log file format did not appear to be compatible with these tools and despite several attempts I could not get any reasonable output from any of them. After another search for possible tools I came across an integrated log file analysis package called Sawmill[7], which at least with the Windows version, included everything necessary to analyse and report on the concatenated Out of Spec log file.

N.B. Correlation information is included throughout the report as hyperlinks.

References:

1 http://www.vmware.com/
2 http://www.mysql.com/downloads/index.html
3 http://www.silicondefense.com/software/snortsnarf/
4 http://www.giac.org/practical/Brandon_Newport_GCIA.zip
5 http://www.dpo.uab.edu/~andrewb/snort/snort_sort.html
6 http://home.earthlink.net/~ckoutras/ws2h111.zip
7 http://www.sawmill.net/features.html