Global Information Assurance Certification Paper

GIAC GCIAv3.3 Practical

John Ruiz

Submission date: July 14, 2003

Part 1 – Network Pattern: Understanding Covert channels

Part 2 – Network detects

Part 3 – Analyze this

Introduction

Covert channels are a topic in the realm of Network Security that has taken a backseat to the viruses, trojans, and various exploits that have taken prominence in the minds of security professionals everywhere. Covert channels are still and always have been a serious threat to the privacy, integrity, and confidentiality aspects of any organization's systems and networks. The goal of this paper is to give the IDS analyst a foundation in understanding covert channels with an emphasis on how protocol exploitation factors into their operation. First, an introduction to covert channel concepts is presented, followed by protocol exploitation strategies an attacker could use when employing a covert channel. We will then focus on some of the covert channel tools and attacks. Finally, the Shared Resource Matrix methodology for the discovery of potential covert channels is also discussed.

Covert Channel Concepts

The average IDS analyst examines countless connections, sessions, and data transfers each day, and for him to discern a covert channel from normal communications requires a firm grasp on what constitutes and defines a covert channel. There are many iterations of the definition of a covert channel; comprehensively, it can be defined as an "illicit means of leaking sensitive or private information through system global variables that usually are not part of the interpretation of data objects in the security model."[2](Shiuh-Pyng Shieh ,1999). In other words, information is transmitted between computer systems in a format that is not consistent with the design of the system. Some aspect of that system, whether it is a network protocol or an operating system is usually manipulated for the purposes of leaking information, not what it was originally intended for. Some of the easiest aspects of a networked computer system to manipulate are the specific protocols that are being used on that system. It is possible for a protocol to be altered such that one or more of its controlling aspects become data objects, used to illegitimately and surrpetiously carry data in unassuming or benign packets. For an analyst to recognize these illegitimate data objects, he must know how information is stored in valid data objects. This can be dependent on: 1) the design and intended use of the system and 2) what an organization's security policy allows as valid or invalid; thus what qualifies as a covert channel can depend on the security policy and is not based purely on technological factors. Basically, a covert channel exists contrary to the design of the computer system, policy-wise and/or technology-wise, although the focus of this paper will be primarily on the technology aspect.

One of the most important factors a covert channel takes advantage of is the use of a global variable. Global variables are data objects that store or signal data; they can be altered or viewed by a process running on a local or sometimes even remote machine. What does a global variable look like? It could be some mundane variable in your Linux kernel used to keep track of disk reads. It could also be the TCP initial number sequence field used to keep track of TCP/IP communications. It could even be a CPU cycle! Chances are if you can signal or store a bit in it, it could be used as an illicit means of data leakage. Global variables are what distinguish the types of covert channels. Covert channels can be categorized into two basic types: storage channels and timing channels.

The global variable(s) in a storage channel are the attribute(s) of a shared resource or system variable which can potentially be used as a vehicle to transfer information, data is implied to be stored on some medium(the data must somehow be encoded into the global variable); the global variable can be altered by a system call (operating system level), programming function or method (executable level), or application (user level). Regardless of which level this shared global variable can be changed, a storage channel is only fully realized if this variable can be viewed or referenced by another process (and the enclosed data subsequently decoded). See figure 1.1 [2](Shiuh-Pyng Shieh,1999). Insofar as the schemes used to interpret the data stored in the global variable, this is up to the covert channel designer to decide. For example, if the global variable the attacker decides to use is a file, more specifically the file-lock attribute of a file, a lock on the file would signal a "1" whereas an unlocked file would signal a "0" in a pseudo binary data signaling scheme. Any one of a number of schemes can be implemented as long as attackers are creative.

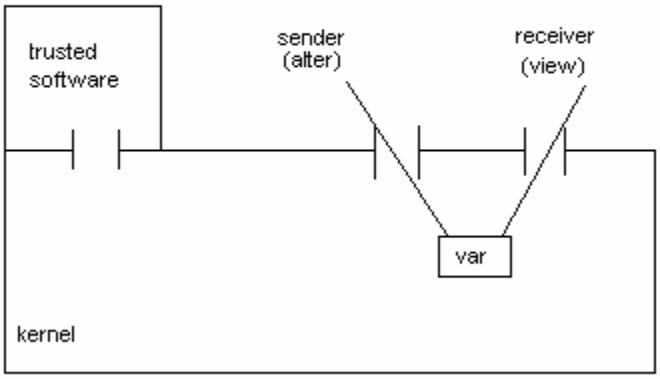


Figure 1.1 - A covert storage channel

The other type of covert channel is a timing channel. Timing channels use timing or ordering relationships for accesses to shared resources as the global variable. This definition of a global variable differs considerably from the data-container type described earlier. Data isn't exactly stored in a memory location as implied by a storage type channels. You'll be hard pressed to find bits and bytes stored in a timing channel. Instead, the bits and bytes are signaled by timed or ordered accesses to a shared resource such as a CPU. Usually this requires cooperation by both subjects of a covert channel in referencing a realtime clock. For example, say that two subjects both have access to and reference the same timer/clock, and that subject A can monitor the amount of time subject B runs process Z. In yet another pseudo binary signaling scheme, if subject B runs process Z for more than 10 seconds this would signal a "1", if subject B runs process Z for less than 10 seconds than this would signal a "0". The shared resource in this particular instance would be the CPU and the global variable would be the signaling scheme and amount of CPU time taken to run that process.

One obvious characteristic both types of covert channels share is that both involve a sender and receiver. Usually, there is a subject with access to certain information and another subject that wishes to have unauthorized access to that information. The sender

and receiver can be involved in one of two relationships when using a covert channel. If both subjects use a noiseless covert channel, this means that they communicate using a shared resource that is exclusive only to them. If the subjects use a noisy covert channel, this means that they communicate using a shared resource that is not exclusive to them and is available to other subjects as well. Noisy channels are harder to use since the shared resource can potentially be modified by non-covert channel subjects thus making it more difficult for the sender and receiver to distinguish extraneous information from the actual covert channel information flow.

Storage channels are the more "popular" type of covert channel. Storage channel software and tools are apparently more accessible and numerous than timing channel software, thus the focus of the remainder of this paper will be on covert storage channels, unless otherwise noted.

How can an analyst determine if there is a potential covert channel on their monitored system? There are several conditions that must be met for there to be a possible covert channel [2](Shiuh-Pyng Shieh,1999):

1. The sender and receiver of the covert channel have the potential to communicate in the system.
2. Communications between the sender and receiver is not allowed under the security policy
3. A global variable exists in the system so as to be accessible by the sender and receiver.
4. The sender must be able to alter the value of the global variable.
5. The change made to the global variable must be detectable/observable by the receiver.
6. The sender and receiver must be able to synchronize their operations so that information flow can take place.

Ultimately, there must be some condition in the operation of the system that must be met, that would initialize and enable information flow through the channel. To verify the existence of an actual covert channel, a real-time scenario must be constructed to fulfill that condition.

Protocol Exploitation

Protocol exploitation in the context of Internet and network security, is the concept of taking advantage of the protocols and standards (or non-standards) that regulate data transmission between computers, and manipulating those protocols deliberately for devious and malicious purposes; in the case of storage covert channels this would be to leak and carry data. Protocol exploitation is often incorporated into the covert channel strategy and design since it is a technique that is effective in NIDS evasion. Also, what better way to be covert than to manipulate protocols and still have the resultant traffic appear legitimate, unassuming, and most importantly undetectable by your standard NIDS? To hide in plain view, so to speak. Remember those crazy 3-D stereogram pictures that people had to cross their eyes for, just to see an image that popped out of the page? Well, they wouldn't have known an image was in that picture

unless by chance or if someone actually told them, this is similar to the case with covert channels. Unless, the analyst knows what to look for to "see" the covert channel, chances are they will miss it. Although there are other techniques attackers can use to covertly transmit information (such as image steganography or even encryption), we will concentrate on the covert channels that exploit the Network and higher layer protocols (pertaining to the OSI model).

Protocol exploitation has become a key component of covert channels for several reasons. For one, covert channels which exploit Internet protocols (esp. the transport and network layers) tend to have less noise introduced into them than covert channels which use some other aspect of a computer system such as file attributes or CPU cycles. First, many of the shared resources in a protocol based covert channel, namely the protocol-controlling fields, options, etc., are exclusive to and modifiable mostly by the peers using that particular protocol. Although there are intermediate systems such as routers and firewalls that have the ability to modify certain fields (TTL in IP for example), they mostly reference these fields to route traffic to their destination. Thus, many of the values in protocol-controlling fields need to remain static and intact in order to maintain integrity of communications and are modifiable only by the communicating peers. All in all, there is not much in the way of introducing extraneous information in protocol-based covert channels.

Protocol exploitation is also useful in NIDS evasion. NIDS evasion techniques rely heavily on the ambiguous behavior of Internet protocols which can be easily exploited by an attacker for use in a covert channel. NIDS are susceptible to this ambiguity in three different ways[3](Paxson, Vern & Handley, Mark 2001):

1. The NIDS may not perform complete analysis of the entire range of behavior for a particular protocol. (i.e. some NIDS will not perform IP packet reassembly), therefore it cannot detect activity that may have a particular behavior that is not analyzed and processed by the NIDS.
2. Unless the NIDS knows an end systems protocol implementation, it may not know how that end system will handle peculiar or unusual traffic, since protocol implementations can vary considerably.
3. The NIDS may never know if traffic will reach its intended destination, if it does not know the topology of the network.

This ambiguity will show up among different implementations of a protocol; erroneous or exceptional conditions are handled in an unstructured manner and some features or facets of a protocol are used inconsistently (or sometimes not at all). For example, in some TCP/IP implementations when receiving overlapping fragments, the TCP/IP stack will give preference to data in preceding fragments when reassembling a packet; other implementations will include data from the succeeding packets. In a covert channel, an attacker could use this inconsistency to his advantage by placing data either at the end of beginning of overlapping fragments, rendering it undetectable by NIDS that perform incorrect (or non-existent) packet reassembly. There really are too many examples of protocol variation to list. Ultimately, each protocol will have its own idiosyncrasies, flaws, and weaknesses. An analyst will have a difficult time; how does one detect traffic that is evasive in nature? An attacker can examine these protocol

weaknesses and inconsistencies in formulating a strategy in the design and use of his covert channel.

Covert Channel Design

So where does an attacker start out when creating a covert channel? A good initial strategy would be to try to verify if all the conditions for the existence of a possible covert channel are true and/or applicable to the target system (assuming it is on a remote network over the Internet):

1. The sender and receiver of the covert channel have the potential to communicate in the system.

This addresses the existence characteristic of covert channels, there must be a path along which information can be transmitted. The Internet is basically a public network, every host has the potential to communicate with almost any other host as long as a route exists between them. It is relatively easy to determine if this condition is true. Discovering if a host is available is fairly straightforward using tools such as traceroute or ping. However, if the target system is protected by network perimeter defenses (such as a basic packet-filter) that filter certain protocols or traffic, more advanced techniques and tools (for example, nmap, hping, firewalk) can be used to discover alternate pathways; Subsequently, a covert channel could be created by exploiting an alternate protocol or application that is not blocked, unless the original exploited protocol was absolutely necessary for the attacker's purposes.

2. Communications between the sender and receiver is not allowed under the security policy (when relating to open systems like the Internet though, this condition by and large is only partially met due to lenient, poorly-enforced or non-existent security policies).

A formal security policy may not be in place for a particular organization, but since the purpose of a covert channel is to leak sensitive or private information, for the most part this violates the purpose and use of any system, unless it is a honeynet. It is difficult to ascertain what the security policy (assuming one exists and is enforced) of an organization. What is relevant is that the activity of a covert channel usually is illegitimate or invalid by that organization's standards, formal or informal. If there is no security policy, the subjects using the covert channel must be in different protection domains.

3. A global variable exists in the system so as to be accessible by the sender and receiver.

With the various protocols being used on the Internet, attackers have a multitude of ways to hide information in the fields, flags, and options used to control these protocols. Basically there are many, many potential global variables available for use in a covert

channel; all an attacker has to do is choose a protocol to carry the covert channel over and discover which global variables would be best suited to carry data.

4. The sender must be able to alter the value of the global variable.

There are a wide variety of easily attainable tools, APIs and other programming interfaces that have made it considerably straightforward to develop network/protocol-manipulating programs. For example, there is the API libnet which allows a programmer to inject manipulated packets onto a network. Any attacker with considerable programming experience should easily be able to do this.

5. The change made to the global variable must be detectable/observable by the receiver.

In order for the receiver to communicate using a given protocol, they must have the software and/or applications to handle the protocol that contains the altered global variable. That way the receiving process has to at least "see" the altered variable even if doesn't completely process that particular variable in its protocol implementation (instead it will be processed by the attacker's covert channel software). This condition is relatively more difficult to confirm than the others. The sender must have in depth knowledge of the receiver's system, to know that the receiver has the necessary software to process the global variable.

6. The sender and receiver must be able to synchronize their operations so that information flow can take place.

Most protocols have built in procedures for coordination between the sender and receiver (for example, TCP relies on acknowledgements). In fact, it is essential that the core operability of that protocol be intact on both end systems and intermediate systems, so that communications between the covert channel sender and receiver may take place. The sender and receiver, however, must still coordinate the covert channel so that both will know when the information flow begins and ends. They must also handle the encoding and decoding of the data stored in the global variable. The availability of the covert channel must also be considered; is it readily available at all times? Or must the end systems be alerted so that information flow can take place? The attacker must incorporate these mechanisms into his covert channel.

If an attacker is confident that the above conditions can be met, the next logical step in the design of a protocol-based covert channel is to choose a protocol to carry it. It would be futile to do an exhaustive analysis of every protocol just to see if it is a suitable candidate. Knowledge of a protocol's specifications and its ambiguities is helpful and selections can be narrowed down based on a few criteria:

1. The manipulated version of the protocol/application must not seriously affect or be seriously affected by network or systems operations. If it did, there is the possibility that the resultant traffic would exhibit overt anomalous characteristics

that could be observed by the IDS analyst; also, there is the possibility that the covert channel would drop packets in transmission. The ambiguity in the way some protocols work contributes to this factor, since ambiguity plays a role in NIDS evasion.

2. The global variable chosen to carry the data must have enough bits in it to handle the "alphabet" the attacker has chosen. For example, if the attacker wanted to leak a text file, one English character at a time, he would have to choose a global variable that could accommodate 26 letters or at least 2^5 bits. The attacker must also design an encoding/decoding scheme for the data enclosed in the global variable.

3. The signal-to-noise ratio must be acceptable. This factor has more to do with noisy covert channels than noiseless covert channels. The question is, how much noise can the covert channel tolerate before the information being sent becomes garbled and unreadable by a receiving system. Take for example, a covert channel that uses UDP as its carrier; UDP does not have any mechanisms to ensure that sent packets will reach their destination. Inevitably, some packets will be lost, regarding a covert channel, how many packets can be lost before the covert channel loses its effectiveness?

4. The covert channel must have sufficient privileges/permissions to operate on the target system. For example, if the target system was a Linux box, would the covert channel software have to be run with root privileges in order to communicate to system on the other end of the channel?

Probably one of the most crucial aspects of covert channels an attacker must consider and incorporate into its design is the bandwidth. Bandwidth is how rapidly information can be sent using the covert channel. No protocol should have a problem handling the bandwidth requirements of a covert channel since covert channels tend to send only bits of information at a time and unless there is a lot of congestion on the intermediate networks, it should take only a few moments for a packet to reach its destination. However, the purpose of a covert channel is to remain covert, so how fast or slow should packets be sent over the covert channel should be taken into consideration. Send packets too fast and risk being detected; on the other hand, send packets too slow and the covert channel becomes virtually useless. The right balance must be found.

The attacker must also decide what *type* of information he wants to remain covert. There are several aspects of the covert communication that the attacker can hide which include[1](Caloyannides, 2002):

1. The content of the communication.
2. The source of the communication.
3. The identity of the intended recipient.
4. The fact that the communication is occurring in the first place.

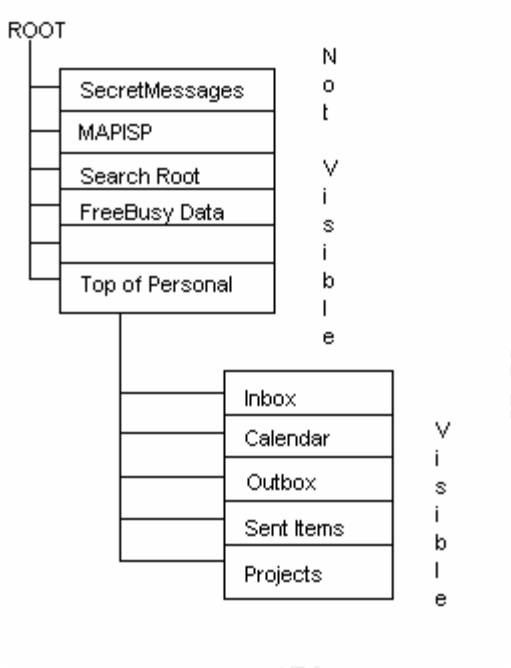
Depending on which aspect(s) the attacker decides to hide will affect the level of complexity of the channel, which doesn't make it easier for the attacker, nor the analyst. It is no easy task for the attacker to design a covert channel, there are many, many factors he must take into consideration in order for the communications to be covert, remain manageable, and operate within the constraints of the target system.

Actual Covert Channels

Covert channels can be implemented at almost any layer of the OSI model. Whether an attacker chooses an application layer-based or transport layer-based channel depends on a number of factors including their design strategy and the architecture of the network and its systems(perimeter defenses, software architecture). The purpose of this section is to present a few "real" and/or potential covert channels in the context of protocol exploitation; these occur at different layers and show how covert channels are not limited to just any one layer.

Bunratty Attack

The Bunratty Attack is a potential application layer covert channel that takes advantage of Microsoft's Messaging API (MAPI) and is an excellent example of how a little knowledge of a simple API can be used in a covert channel. MAPI has several features and capabilities built into it that aren't readily apparent to the end user that can be exploited to secretly transmit messages in standard MAPI format (perhaps this covert channel wants to hide the fact that communications are occurring in the first place). An understanding of the MAPI protocol is necessary to comprehend how this all works. In the MAPI client, the Exchange Inbox, users have access to a message store, *Personal Folders,* that contains several folders (Inbox, Outbox) that the user sees as the root of the directory architecture. In reality, *Personal Folders,* is one several root level folders, of which the other folders are not visible to the end user. See accompanying figure[6](Gallo,2001):



The reason for implementing these hidden folders was so that messages and other data could be stored so that they would not be accidentally damaged or seen. How are these hidden folders useful to the attacker who wants to exploit MAPI for use in a covert channel? Well, the problem is, that these hidden folders make an excellent choice as a global variable in a storage type channel; no special encoding or decoding is needed since these folders store standard email messages, also, the attackers isn't really limited by the amount of bits he can store but rather by the amount of bytes which makes for decent bandwidth. To alter the information stored in these hidden folders, another capability of MAPI must be exploited. MAPI uses a *routing table* to determine which folders messages should be delivered to. Each message has a property known as the *message class.* Depending on what message class a message has associated with it, MAPI will look at the message class, look up which folder that message class is associated in its routing table, and proceed to deliver the message to the appropriate folder. For example, a message with the message class IPM will be sent to the Inbox folder. How can all these features be exploited in a covert channel? Let's take for example an attacker who is skillful in writing MAPI applications and has written a MAPI-based covert channel. On the target system, his software has created a hidden folder called SecretMessages that is in the root level directory. Also, the software has modified the MAPI routing table on the target system so that any messages with message class "MSG.Secret" would be sent to the SecretMessages folder (it doesn't help that the message doesn't pass through Inbox first). If the software was capable, these messages could contain commands for the remote control of that system or it could tell it to send back email messages containing critical or sensitive data. In the end, we have a covert channel based on a simple API, sends data in a format that look like standard email messages, and is almost totally invisible to the end user.

Covert_tcp

Covert_tcp is a proof of concept covert channel that works at the transport and network layers; it is a good example of the availability of global variables and how to creatively use them. In particular, it uses several fields in the TCP/IP header as global variables for the transmission of ASCII data. These fields are:

1. IP packet identification field
2. TCP initial sequence number field
3. TCP acknowledged sequence number field

These particular fields were selected because they were less likely to be altered by perimeter devices/software such as packet filters. The result is a covert channel that is not seriously affected by network or systems operations, which lends itself well towards fidelous and reliable communications. They also hide the content of communication quite well, masquerading as packets in initial connection requests and established connections.

However, these fields were not meant to carry bytes and bytes of data (they weren't even meant to carry data), they were only meant to keep track of state which requires only a few bits; thus only bits of data are available for the transmission of data. This is enough to accommodate the alphabet of covert_tcp which is in ASCII. As a result, covert_tcp's bandwidth is only a trickle, transmitting data one ASCII character per packet at a time. In the actual encoding/decoding scheme, the value in these fields is divided by some number (depending on if it the packet it is contained in is an IP ID, TCP ISN or TCP ACK) to obtain an ASCII value. For example, when the IP ID field is used as the data object for transmission, the receiver upon receipt of the packet will parse the IP ID field, obtain the value stored in it and divide that value by 256 to get the numerical representation of some ASCII character, i.e. sending the word "HELLO" would result in the following sequence of packets (these sample packets taken from source[8](Rowland)):

Packet One:

```
18:50:13.551117 nemesis.psionic.com.7180 > blast.psionic.com.www: S
537657344:537657344(0) win 512 (ttl 64, id 18432)

Decoding:...(ttl 64, id 18432/256) [ASCII: 72(H)]
```

Packet Two:

```
18:50:14.551117 nemesis.psionic.com.51727 > blast.psionic.com.www:
S1393295360:1393295360(0) win 512 (ttl 64, id 17664)

Decoding:...(ttl 64, id 17664/256) [ASCII: 69(E)]
```

Packet Three:

```
18:50:15.551117 nemesis.psionic.com.9473 > blast.psionic.com.www: S
3994419200:3994419200(0) win 512 (ttl 64, id 19456)

Decoding:...(ttl 64, id 19456/256) [ASCII: 76(L)]
```

Packet Four:

```
18:50:16.551117 nemesis.psionic.com.56855 > blast.psionic.com.www:
S3676635136:3676635136(0) win 512 (ttl 64, id 19456)

Decoding:...(ttl 64, id 19456/256) [ASCII: 76(L)]
```

Packet Five:

```
18:50:17.551117 nemesis.psionic.com.1280 > blast.psionic.com.www: S
774242304:774242304(0) win 512 (ttl 64, id 20224)

Decoding:...(ttl 64, id 20224/256) [ASCII: 79(O)]
```

Packet Six:

```
18:50:18.551117 nemesis.psionic.com.21004 > blast.psionic.com.www:
S3843751936:3843751936(0) win 512 (ttl 64, id 2560)

Decoding:...(ttl 64, id 2560/256) [ASCII: 10(Carriage Return)]
```

Ostensibly, these packets are part of an established TCP/IP communications session, yet no session had ever been established.

The other methods covert_tcp uses are also based on the premise of exploiting TCP/IP in terms of global variable usage, except that they differ in their strategies in getting packets to their intended destination. When the TCP initial sequence number field is used as the global variable, it also transmits ASCII characters one packet at a time, although its 32-bit size makes it possible to send even larger amounts of data. Using this method, all the data is sent in packets with the SYN flag set, making it appear as if multiple connection attempts are occurring on the network. Once a packet is received, the TCP ISN field is parsed and the value in it is divided by 16777216 to obtain the ASCII numerical representation. No byte ordering is used when encoding data into the ISN field, therefore making it look "natural".

Analysis Strategies

What should be of concern to the IDS analyst is that protocol exploitation-based covert channels are intentionally crafted in such a way as to appear legitimate or part of normal traffic.Unless, the analyst wants an extremely high false positive rate, he cannot just employ signatures that look for these covert channels, since the packets they leave look like every other packet going across the network, normal and unassuming. For that reason, instead of looking for characteristic imprints in packets, the IDS analyst must search at another level. Remember, covert channels are based on sharing resources and

the creative ways in which those resources are used to transmit information; covert channel detection is based on this principle.

The Shared Resource Matrix Methodology (SRM) is a technique for the identification of potential covert channels. It can be used to identify both storage and timing channels at several levels: English requirements, formal specifications, and implementation code. The basis for the SRM lies in the identification of the shared resources in a particular system, their attributes, and relations among processes that control them. This information is then used to generate a matrix graph of those relations, which the analyst must examine for potential covert channels.

Applying the SRM in analysis is a two part process. The purpose of the first part is to generate the matrix to be analyzed. This involves several steps:

1. Identify the shared resources and their attributes; these will label the rows of the matrix.
2. Identify the system operations that can reference and/or modify the shared resources; these will label the columns of the matrix.

Side note: Depending on what level the SRM is applied to, the descriptions and names on the matrix will vary accordingly. For example, if the level were at English requirement, than labels might be "file" or "disk" as opposed to "integer" or "string", which might be at the implementation code level.

3. Fill in the matrix by determining which system operations can reference and/or modify which shared resources/attributes. Some operations can affect one or more shared resources/attributes.
4. For any cells not filled in, use transitive closure to reveal indirect relations.

Sometimes, there is a relationship among operations which allows for the indirect viewing of resource attributes. These relationships can be defined by transitive closure. Transitive closure can be defined by the following formula:

If x R y and y R z, then x R z.

Basically, if there is a relationship between x and y, and there is a relationship between y and z, then there is also a relationship between x and z. When applied in the Shared Resource Matrix, this helps in the identification of complex covert channels where certain attributes can be indirectly referenced or modified by several different operations. The following is an example of a generic matrix:

Resource Attribute	System Operation
A	M
B	R
C	R,M
D	R
E	M
F	R

Where R = Reference/View the resource and M = Modify the resource
It is common for there to be more than one system operation (and thus more than one column). However, for the sake of simplicity, only one system operation was included in our generic matrix.

Once the matrix is generated, the second part involves actually identifying potential covert channels. The first step would involve identifying the shared attributes that can be modified as well as referenced by the either the same or different system operations. Remember, covert channels are based on global variables that can be referenced by a receiving subject and modified by the sender; both need to have access to the same shared resource/attribute in order to successfully communicate, this is why it makes sense to identify attributes that can be referenced and modified by a unique process.

The 6 conditions for a potential covert channel must also be present. First, conditions 1-5 must be fulfilled, any attribute which does not fulfill condition 1-5 can be ignored as being a potential covert channel. The next step is to find a scenario that fulfills the last condition, which is "The sender and receiver must be able to synchronize their operations so that communications can take place". This part requires a bit of creative thinking on behalf of the analyst. For example, to initiate communications over the channel, does the sender first need to send UDP packets to port 123 on the receiver which then starts the process that has access to the attribute being used as a global variable? There are numerous possibilities as to how covert channel communications can be initiated and synchronized which could cover a whole paper. The purpose is to find a weak link in the system.

Once potential covert channels are identified in the generated matrix, they can classified into several different types[4](Kemmerer,2002):

1) A legal channel already exists between the two communicating processes, there is no cause for alarm.
2) The covert channel cannot communicate any useful information.
3) The sending and receiving processes are the same.
4) There is a genuine covert channel.

If the analyst believes that they have found a genuine covert channel, what do they do next? What the analyst should be concerned with is how "effective" the covert channel is; there are many questions to be addressed. Mainly, how much data can it carry, what is its bandwidth (bits/sec? bytes/sec?), and what steps can be taken to limit its bandwidth or eliminate it entirely. The reason we bring up bandwidth is that in some cases a (potential) covert channel cannot always be eliminated. In that situation, steps are taken to limit the bandwidth of the channel so much, so that it becomes useless. Elimination of covert channels is another story, how many attributes can be used as global variables? Which processes are they associated with? In the case of protocol-based covert channels, the easiest way to eliminate a covert channel would be to shutdown the protocol, in almost all situations this is not feasible.

References

1)Caloyannides, Michael A. "Overview of Covert Communications through networks" Nov. 2002
URL: http://minbar.cs.dartmouth.edu/greecom/ejeta/second-issue.php?download=ejeta-2002.05.10.15.51.05.pdf

2)Shiuh-Pyng Shieh (1999) "Estimating and Measuring Covert Channel Bandwidth in Multilevel Secure Operating Systems" Journal of Information Science and Engineering January 1999, pp.91-106

3)Paxson, Vern & Handley, Mark (2001) "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-end Protocol Semantics" IN: Proceedings of 10th Unix Security Symposium August 2001

4)Kemmerer, Richard A. (2002) "A Practical Approach to Identifying Storage and Timing Channels: Twenty Years Later" IN: Proccedings of 18th Annual Computer Security Applicationss Conference Dec. 9-13,2002 pp.109

5)Moskowitz, Ira S. & Kang, Myong H. (1994) "Covert Channels – Here to stay?" IN: Proceedings of Compass 1994 Gaithersburg, MD IEEE Computer Soc. Press. pp.235-243

6)Gallo, Vince (2001) "The Bunratty Attack" Information Security Bulletin Volume 6. Issue 5 June 2001 pp.29-34

7)McHugh, John (2001) "Covert Channel Analysis" Navy Handbook for the Computer Security Certification of Trusted Systems Feb 21, 2000

8)Rowland, Craig H. "Covert Channels in the TCP/IP Protocol Suite
URL: http://www.firstmonday.dk/issues/issue2_5/rowland

ASSIGNMENT #2

DETECT #1

```
[**] [1:184:3] BACKDOOR Q access [**]
[Classification: Misc activity] [Priority: 3]
08/24-23:48:27.764488 255.255.255.255:31337 -> 138.97.144.53:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
[Xref => arachnids 203]

08/24-23:48:27.764488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
255.255.255.255:31337 -> 138.97.144.53:515 TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 2B 00 00 00 00 0E 06 FF F7 FF FF FF FF 8A 61  .+.............a
0x0020: 90 35 7A 69 02 03 00 00 00 00 00 00 00 00 50 14  .5zi..........P.
0x0030: 00 00 06 1F 00 00 63 6B 6F 00 00 00              ......cko...

[**] [1:184:3] BACKDOOR Q access [**]
[Classification: Misc activity] [Priority: 3]
08/24-00:36:57.784488 255.255.255.255:31337 -> 138.97.82.198:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
[Xref => arachnids 203]

08/24-00:36:57.784488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
255.255.255.255:31337 -> 138.97.82.198:515 TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 2B 00 00 00 00 0E 06 FF 66 FF FF FF FF 8A 61  .+.......f.....a
0x0020: 52 C6 7A 69 02 03 00 00 00 00 00 00 00 00 50 14  R.zi..........P.
0x0030: 00 00 43 8E 00 00 63 6B 6F 00 00 00              ..C...cko...

[**] [1:184:3] BACKDOOR Q access [**]
[Classification: Misc activity] [Priority: 3]
08/24-00:43:45.764488 255.255.255.255:31337 -> 138.97.240.167:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
[Xref => arachnids 203]

08/24-00:43:45.764488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
255.255.255.255:31337 -> 138.97.240.167:515 TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 2B 00 00 00 00 0E 06 FF 85 FF FF FF FF 8A 61  .+.............a
0x0020: F0 A7 7A 69 02 03 00 00 00 00 00 00 00 00 50 14  ..zi..........P.
0x0030: 00 00 A3 AD 00 00 63 6B 6F 00 00 00              ......cko...

[**] [1:184:3] BACKDOOR Q access [**]
```

```
[Classification: Misc activity] [Priority: 3]
08/24-01:47:00.784488 255.255.255.255:31337 -> 138.97.197.230:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
[Xref => arachnids 203]

08/24-01:47:00.784488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
255.255.255.255:31337 -> 138.97.197.230:515 TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 2B 00 00 00 00 0E 06 FF 44 FF FF FF FF 8A 61  .+.......D.....a
0x0020: C5 E6 7A 69 02 03 00 00 00 00 00 00 00 00 50 14  ..zi..........P.
0x0030: 00 00 D1 6B 00 00 63 6B 6F 00 00 00              ...k..cko...
```

1. Source of Trace

The trace was obtained from the incidents.org website: http://www.incidents.org/logs/Raw. The specific log file used was 2002.7.24.

2. Detect was generated by:

The alerts and subsequent packet dumps were generated by a Snort IDS v1.8.3 running on Windows XP Professional. All commands were issued at the DOS command prompt. The following command was used to generate the alert.ids file and tcpdump output file:

```
snort -r 2002.7.24 -c C:\snort\snort.conf -l C:\Snort\logs.
```

This resulted in the alert.ids file and tcpdump.log.1049497140 files. The tcpdump file was converted into readable ASCII format and written to a text file using the following command:

```
snort -vdeX -r tcpdump.log.1049497140 > temp.txt
```

The rule which generated the detect is the following and is included in the default backdoor.rules files with Snort:

```
alert tcp 255.255.255.0/24 any -> $HOME_NET any (msg:"BACKDOOR Q access"; flags:A+; dsize: >1;
reference:arachnids,203;
 sid:184;  classtype:misc-activity; rev:3;)
```

The detects were logged by this rule because they matched the 3 main criteria of the rule which were:

1)source IP address of 255.255.255.0 w/ any tcp port number going to the subnet specified by $HOME_NET w/ any tcp port number

```
alert tcp 255.255.255.0/24 any -> $HOME_NET any
```

2)TCP flags set to at least ACK and any other flag

```
flags:A+;
```

3) a datagram size greater than 1 byte.

dsize: >1

More specifically, all three detected packets had the same: source IP, 255.255.255.255 with TCP port 31337; TCP flags combination, ACK + RST; and datagram length of 43 bytes.

3. Probability the source address was spoofed

The source address is definitely spoofed. 255.255.255.255 is a special type of IP address that is known as the limited broadcast address. This address is used when a sender wants to direct a packet to all hosts on the local segment, this address is not routable and cannot be assigned to an individual host. Notice that all of the detected packets had TCP flags RST-ACK set; a packet with these flags set is sent in response to a SYN packet which has solicited a connection to a closed port. This suggests that the destination hosts in all the packets had originally tried to initiate a connection from port 515 to port 31337 on 255.255.255.255. Since a host cannot have the address 255.255.255.255, it is not possible for 255.255.255.255 to send back a response with RST-ACK to indicate a service is not available; also, the sequence number for all the packets is set to 0 as well as the ACK. There is some heavy TCP/IP protocol manipulation occurring.

4. Description of Attack

The rule which generated the detect suggests that this activity is related to the Q Trojan, based on its classification (Snort ID: 184, arachnids: 203). From the traces, it appears as if several print servers had been rejected from establishing connections to 255.255.255.255 on port 31337. Port 31337 does not fall within the range of well known services (ports 1-1023), but port 515 does, it is listed as the printer port which is used by the lpd daemon on unix systems. Did several different hosts from the 138.97.0.0/16 network attempt to initiate communications from the same well-known port to an invalid IP address on an ephemeral port? This is unlikely and uncharacteristic of a print server. There are several things to note when first looking at the packets and their pattern. The source IP and TCP port number do not change and always remain as 255.255.255.255:31337. The destination IP addresses change with each packet but the port numbers remain the same (TCP port 515). This is characteristic of a scan for port 515. Therefore this detect can best be described as some sort of specialized scan. Perhaps someone trolling the 138.97.0.0/16 network for a host compromised by the Q Trojan.

5. Attack Mechanism

Initially, it appears as if this scan is useless. The nature of the packets is not conducive to returning a response to the attacker. The source address is not routable and probably spoofed, and the TCP flags used would cause the packets to be silently

discarded when received by a host's TCP/IP stack. It does not seem any sort of valid connection attempt is occurring; why attempt to connect to port 515 if no response can be elicited? Well, if this scan does not rely on TCP/IP for a response, than perhaps it relies on something else, like a response from the Q Trojan software previously installed on a system (this would be the server version, known as qd). The Q Trojan (qd) is activated by raw packets; raw packets are basically packets that are processed or created by an application other than a host's TCP/IP stack. Recalling the nature of the packets, they definitely appeared NOT to come from a normal TCP/IP stack, exhibiting characteristics that they were created by an application using raw sockets, possibly a client version to the Q Trojan (known as qs). qs sends raw packets one-way to a Q-compromised host to signal qd to spawn a Q secure shell or Q session bouncer that would redirect a session to another host. Based on this further speculation, the classification of this activity as a "specialized scan" as mentioned in the attack description above, can be upgraded and can now be described as a "Q Trojan activation scan". Another strange thing about the packets is their payload, all of the packets payload are virtually the same and contain the characters "cko". One would have to know the details of the Q Trojan to know if and how it processed this string. These packets could be the result of a modified version of qs. Pre 2.0 versions of qs did not use encryption when sending their activation payload (they usually contained information such as which ports to connect to). The payload of these packets appears to be in plain text, so we can surmise that if indeed this is the Q Trojan client, a pre 2.0 version is being used.

Ultimately, though it seems as if the Snort signature used to capture this traffic is too weak to confirm it is actually sent by the client side version of the Q Trojan, qs. The Q Trojan can use TCP, UDP, or ICMP as the carrying protocol; source IP and port number can be randomly generated and is not just limited to 255.255.255.255. However, the traffic is unusual enough in that a scan is definitely occurring, and it exhibits characteristics of the Q Trojan (one-way raw TCP/IP packets, irrelevant destination TCP port numbers).

6. Correlations

This type of activity has also been seen by several members of the incidents.org mailing list. In particular Patrick Cheong Shu Yang posted his detect of the attack on May 3, 2001 (http://securityfocus.com/archive/75/181909):

I have also seen the same potential intrusion from our Snort logs as follows:-

```
11:05:18.603917 255.255.255.255.31337 > xxx.xxx.xxx.xx.515: R 0:3(3) ack
0 win 0
0x0000   4500 002b 0000 0000 0e06 1900 ffff ffff        E..+.....
0x0010   cab9 c914 7a69 0203 0000 0000 0000 0000        ....zi.....
0x0020   5014 0000 cd27 0000 636b 6f00 0000             P....'..cko...
```

Anyone else seen this and can anyone explain what this is?!?!

The traces look identical to the ones I found with the exception of the timestamp being different. Other members have speculated as to exactly what kind of activity this is, but none have really been able to confirm what tools or software were used to generate the activity. Jeff Peterson (http://securityfocus.com/archive/75/182215) speculated that this might be some sort of IRC Trojan scan as he detected this activity whenever he connected to a certain IRC. Jason Storm also speculated that this activity might be some sort of IRC-related Trojan divulging that the IPs he detected the activity on were almost exclusively used for IRC (http://securityfocus.com/archive/75/182245). All in all no-one was able to confirm this activity as being the Q Trojan for certain, but there seemed to be the general conjecture that this was some sort of IRC-related Trojan.

7. Evidence of Active Targeting

It does not seem as if any particular host was targeted in this scan. The IP addresses scanned seem to be random; there is no pattern to the IP addresses scanned as they seem to vary widely and there is no pattern to the time intervals between scans . However, it does seem as if only the 138.97.0.0/16 network was targeted.

8. Severity

Severity is calculated using the following formula:

Severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Criticality: 3 – Since these logs were obtained from www.incidents.org, not much is known about the criticality of the targeted systems. Therefore we will give Criticality an "average" score.

Lethality: 5 – If this attack were successful, an attacker would be able to execute remote commands interactively on the target system. The traffic between attacker and target would be also be encrypted, making it difficult to ascertain what activity the attacker was performing.

System countermeasures: 3 – Since these logs were obtained from www.incidents.org, not much is known about what system countermeasures were in place to protect individual hosts.

Network countermeasures: 1 – First of all, these packets were able to be detected by the IDS. This could mean several things. These packets had to be routed by some router and possibly a firewall meaning that they did not filter out the packets even though they had an invalid source IP. This could signify poor ACL and firewall rules. Also, the target network possibly has "allow all not specifically denied" as part of their access control policy. There is no reason why packets with destination TCP port 515 should be able to get into the network, unless specifically allowed (there really isn't any reason why someone from outside your network should be able to print on your network).

Severity = (3 + 5) – (3 + 1) = 4.

9. Defensive Recommendations

It is apparent that the ACL and firewall rules do not block on invalid or unroutable IP addresses. This is one of the first things that should be addressed, ACL and firewalls rules should be added that block on unroutable IP addresses (such as broadcast, and private/reserved IP addresses). Next, if possible, a "deny all not specifically allowed" type policy should be implemented, that way people would have access only to necessary services.

10. Multiple choice test question:

If FIRST sent to a host, which attribute(s) of the following packet is not conducive to eliciting a response from that host?

```
08/24-23:48:27.764488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
255.255.255.255:31337 -> 138.97.144.53:515 TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 2B 00 00 00 00 0E 06 FF F7 FF FF FF FF 8A 61  .+.............a
0x0020: 90 35 7A 69 02 03 00 00 00 00 00 00 00 00 50 14  .5zi..........P.
0x0030: 00 00 06 1F 00 00 63 6B 6F 00 00 00              ......cko...
```

a) Source TCP port number
b) ACK-RST flags set
c) Window size of 10
d) Sequence number of 0

Answer: B

Intrusions.org mailing list questions and post (originally posted June 18, 2003):

Reply to Andre Comer:

How can you tell that this packet was not blocked ? Do you have information on where the sensor was located ? What if the sensor is between the ISP edge router and a firewall and the packet is inbound ?

What information about the packets or patterns can tell us that this packet was detected behind a firewall? No attributes of an individual packet can tell you for sure. However, you can infer from TTLs where the router and firewall nodes are in a network. Depending on what the attacking system is running, TTLs can take on one of several values (varying from 32,64,128, etc). If you are "luckY" and there is a single attacking system hitting many different subnets and services. You can map out where these nodes are by looking at the TTLs, the lesser the value of the TTL from that single attackers IP, the more internal nodes (routers, firewalls) the packet it has passed through. You can get a general idea of how the network looks like, where all the webservers are, dns servers,

workstations. Due to the criticality of the systems you can probably guess where a firewall or IDS should
be, BUT without any sort of network topology on hand, you will never be able to tell.

You seem to beleive that 138.97.0.0/16 is the IP range of the sensor owner. How did you came to that conclusion ?

Parsing through the logs it seemed as if the 130.85.x.x IP addresses were more prevalent than any other IP addressess or subnets, and seemed to be the target (on well known ports) for most of the time. This does not necessarily mean that it is the home network (and where the IDS sensor resides). However, the other IP addressess present in the logs varied very much, it seems too much for any one IP address or subnet to be the home network. If 130.85 was in fact an external network than it would seem really odd that all these different IP addressess and subnets would be scanning 130.85. So do its prevalence the extreme variance in the other IPs/subnets in the logs, chances are 130.85 is the home network.

Reply to Ken Claussen:

If these were indeed constructed packets, which they appear to be as you say, then why choose a port which would likely trigger IDS systems as opposed to say 45680 (Random High Port).

It is possible that this particular activity is due to a script kiddie who obtained some sort of backorifice/Q scanner (or possibly even backorifice/Q itself, poor kiddie) and proceeded to scan for port 31337. The nature of the "scan" itself does not lend well to eliciting a response, possibly an amateur mistake. If this were a sign of an actual infection, I believe a "real" attacker would have made the effort to modify the trojan to listen in on a different port, and the scan would have probably been much more stealthy.

Detect #2

```
[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/07-18:59:15.714488 195.228.243.120:902 -> 46.5.115.153:111
UDP TTL:113 TOS:0x0 ID:9700 IpLen:20 DgmLen:84
Len: 64

06/07-18:59:10.084488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:9379 IpLen:20 DgmLen:84
Len: 64
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 54 24 A3 00 00 71 11 D3 FF C3 E4 F3 78 2E 05  .T$...q......x..
0x0020: 73 99 03 86 00 6F 00 40 96 39 62 38 A4 AD 00 00  s....o.@.9b8....
0x0030: 00 00 00 00 00 02 00 01 86 A0 00 00 00 02 00 00  ................
0x0040: 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0050: 00 00 00 01 86 A5 00 00 00 03 00 00 00 11 00 00  ................
0x0060: 00 00                                            ..
```

[Xref => arachnids 13]
[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/07-18:59:45.084488 195.228.243.120:902 -> 46.5.115.153:111
UDP TTL:113 TOS:0x0 ID:11168 IpLen:20 DgmLen:84
Len: 64

06/07-18:59:10.894488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:9421 IpLen:20 DgmLen:84
Len: 64
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00 .....3....&...E.
0x0010: 00 54 24 CD 00 00 71 11 D3 D5 C3 E4 F3 78 2E 05 .T$...q......x..
0x0020: 73 99 03 86 00 6F 00 40 96 39 62 38 A4 AD 00 00 s....o.@.9b8....
0x0030: 00 00 00 00 00 02 00 01 86 A0 00 00 00 02 00 00 ....
0x0040: 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0x0050: 00 00 00 01 86 A5 00 00 00 03 00 00 00 11 00 00 ....
0x0060: 00 00

[Xref => arachnids 13]
[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/07-18:59:45.904488 195.228.243.120:902 -> 46.5.115.153:111
UDP TTL:113 TOS:0x0 ID:11207 IpLen:20 DgmLen:84
Len: 64

06/07-18:59:12.504488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:9508 IpLen:20 DgmLen:84
Len: 64
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00 .....3....&...E.
0x0010: 00 54 25 24 00 00 71 11 D3 7E C3 E4 F3 78 2E 05 .T%$..q..~...x..
0x0020: 73 99 03 86 00 6F 00 40 96 39 62 38 A4 AD 00 00 s....o.@.9b8....
0x0030: 00 00 00 00 00 02 00 01 86 A0 00 00 00 02 00 00 ....
0x0040: 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0x0050: 00 00 00 01 86 A5 00 00 00 03 00 00 00 11 00 00 ....
0x0060: 00 00

[Xref => arachnids 13]
[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/07-18:59:47.504488 195.228.243.120:902 -> 46.5.115.153:111
UDP TTL:113 TOS:0x0 ID:11289 IpLen:20 DgmLen:84
Len: 64

06/07-18:59:15.714488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
195.228.243.120:902 -> 46.5.115.153:111 UDP TTL:113 TOS:0x0 ID:9700 IpLen:20 DgmLen:84
Len: 64
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00 .....3....&...E.
0x0010: 00 54 25 E4 00 00 71 11 D2 BE C3 E4 F3 78 2E 05 .T%...q......x..
0x0020: 73 99 03 86 00 6F 00 40 96 39 62 38 A4 AD 00 00 s....o.@.9b8....
0x0030: 00 00 00 00 00 02 00 01 86 A0 00 00 00 02 00 00 ....
0x0040: 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0x0050: 00 00 00 01 86 A5 00 00 00 03 00 00 00 11 00 00 ....
0x0060: 00 00

1. Source of Trace

The trace was obtained from the incidents.org website: http://www.incidents.org/logs/Raw. The specific log file used was 2002.5.7.

2. Detect was generated by:

The alerts and subsequent packet dumps were generated by a Snort IDS v1.8.3 running on Windows XP Professional. All commands were issued at the DOS command prompt. The following command was used to generate the alert.ids file and tcpdump output file:

```
snort -r 2002.5.7 -c C:\snort\snort.conf -l C:\Snort\logs.
```

This resulted in the alert.ids file and tcpdump.log. files. The tcpdump file was converted into readable ASCII format and written to a text file using the following command:

```
snort -vdeX -r tcpdump.log.1050732052 > temp.txt
```

The rule which generated the detect is the following and is included in the default rpc.rules files with Snort v1.8.3

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request mountd";
content:"|01 86 A5 00 00|";offset:40;depth:8; reference:arachnids,13; classtype:rpc-portmap-decode;
sid:579; rev:2;)
```

The detects were logged by this rule because they matched the 4 main criteria specified by the rule:

1) Communications were occurring using the udp protocol, with a source IP address specified by the $EXTERNAL_NET variable on any port, going to a destination IP address specified by the $HOME_NET variable on port 111.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111
```

2) The payload of the packet contained certain hex data specified by the content keyword:

```
content:"|01 86 A5 00 00|";
```

3) The payload specified by content started at least 40 bytes into the packet:

```
offset:40;
```

4) The maximum search depth is at most 8 bytes from the search point (the offset):

```
depth:8;
```

3. Probability the source address was spoofed.

The source address in the traces is probably not spoofed. This is probably the case since the nature of a portmap request requires a response. Portmapper is a service that when queried by a host, will tell that host which ports are mapped to their appropriate services. What is unusual though is the source host is contacting the destination host on a well-known port from a well-known port. Most of the time, the source host will communicate from an ephemeral port as set by its IP stack. Portmapper does not require queries from well-known ports and there is no reason why the source should have contacted portmapper from udp port 902 which is a well-known port. Therefore, for the source to have set its source port to 902 would have required manipulation of the UDP headers.

4. Description of Attack

This attack can be best described as an attempt to query the portmap service running on a host for the rpc.mountd service. This is more of a reconnaissance type of activity rather than a direct compromise of the target host. The mountd service is part of the NFS (Network File System) which allows remote hosts to mount partitions from a system and use them as if they were local file systems. The purpose of mountd is to answer NFS clients requests to mount a file system and check if they have the appropriate access permissions to the exported file systems. This type of activity could be indicative of an attacker looking to mount an exportable file system for his own use (storage of warez); or it could be an attacker looking to gain elevated privileges on systems running a vulnerable version of mountd.

5. Attack mechanism

This reconnaissance activity works by sending UDP packets to the target host to port 111, which is associated with the portmapper service. These particular packets are meant to query the portmapper service to determine the port on which the mountd service runs. IF the portmapper service is running and the mountd service is running on the target host, the target host will return to the querying host the UDP port number on which to connect and use the mountd service. The attacker cannot rely on the transport layer protocol for responses since NFS uses UDP which is stateless. The responses and information the attacker needs will come directly from portmapper and mountd; their response depends totally on how they are setup.

6. Correlations

This type of activity has been seen many times by members of the incidents.org mailing list. On June 11, 2001 (http://archives.neohapsis.com/archives/sf/linux/2001-q2/0131.html) Brian Clifton reported this activity in his syslog logs on a system running RedHat 6.2. In his logs he observed multiple requests to the portmapper service, specifically querying for the port on which mountd was running:

Jun 1 14:39:37 linux portmap[27164]: connect from 206.218.166.214 to getport(mountd): request from unauthorized host

Jun 6 23:49:02 linux portmap[20055]: connect from 212.55.157.163 to
getport(status): request from unauthorized host

I was curious as to if I could recreate similar traces on a unix system by querying a portmapper service for mountd and by querying a server for exportable mounts. The following command was issued on a unix system running SunOS 5.8 and the subsequent traces were logged by running a tcpdump (tcpdump –X –vvv –w temp) on a target system:

mount -F nfs 192.168.1.1:/usr/data /tmp

The following packets were logged:

01:42:28.503583 sentry3b.33683 > 192.168.1.1.sunrPc: udp 56 (DF) (ttl 255, id 63804, len 84)
0x0000 4500 0054 f93c 4000 ff11 e89e 8cb9 4b5a E..T.<@.....KZ
0x0010 c0a8 0101 8393 006f 0040 89c2 3eaf 0bda .....o.@..>...
0x0020 0000 0000 0000 0002 0001 86a0 0000 0002.....
0x0030 0000 0003 0000 .....

01:42:43.504758 sentry3b.33683 > 192.168.1.1.sunrPc: udp 56 (DF) (ttl 255, id 63805, len 84)
0x0000 4500 0054 f93d 4000 ff11 e89d 8cb9 4b5a E..T.=@.....KZ
0x0010 c0a8 0101 8393 006f 0040 89c2 3eaf 0bda .....o.@..>...
0x0020 0000 0000 0000 0002 0001 86a0 0000 0002.....
0x0030 0000 0003 0000

These packets will match on newer snort (included in v2.0) rules meant to catch RPC portmap request mountd scans, this signature is more specific, the content portion of the rule has changed:

content:"|00 01 86 A0|"; offset:12; depth:4; content:"|00 00 00 03|"; distance:4; within:4;
byte_jump:4,4,relative,align; byte_jump:4,4,relative,align; content:"|00 01 86 A5|";

The content matches are highlighted in red in the packets. Only a portion of the packets were logged because the default snaplength of 68 bytes was used. This is why we do not see the last hex content search string of 00 01 86 A5 in these packets. When the tcpdump log that the original detects came from are run through a Snort 2.0 ruleset they also alert on "RPC portmap request mountd" on the same traffic, here is a snippet of the alert file:

[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/07-18:59:10.084488 195.228.243.120:902 -> 46.5.115.153:111
UDP TTL:113 TOS:0x0 ID:9379 IpLen:20 DgmLen:84
Len: 56
[Xref => http://www.whitehats.com/info/IDS13]

[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]
06/07-18:59:10.894488 195.228.243.120:902 -> 46.5.115.153:111
UDP TTL:113 TOS:0x0 ID:9421 IpLen:20 DgmLen:84
Len: 56
[Xref => http://www.whitehats.com/info/IDS13]

[**] [1:579:2] RPC portmap request mountd [**]
[Classification: Decode of an RPC Query] [Priority: 2]

06/07-18:59:12.504488 195.228.243.120:902 -> 46.5.115.153:111
UDP TTL:113 TOS:0x0 ID:9508 IpLen:20 DgmLen:84
Len: 56
[Xref => http://www.whitehats.com/info/IDS13]

This substantiates that this activity is almost certainly portmapper scans for mountd.

7. Evidence of Active Targeting

The only host that was targeted by the RPC mountd scans was 46.5.115.153. This destination IP did not trigger on any other alerts nor was there any other alert generated by the source IP 195.228.243.120. Therefore it seems as if these scan were very specific in their target.

8. Severity

Severity is calculated using the following formula:

Severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Criticality: 3 – Since these logs were obtained from www.incidents.org, not much is known about the criticality of the targeted systems. Therefore we will give Criticality an "average" score.

Lethality: 1 – The activity in question will not result in the compromise or elevation of privileges on the target system. However, any positive responses elicited from the target will result in the attacker gaining knowledge about if the mountd service is running. The attacker can then take further action such as trying to mount an exportable partition or finding a vulnerability for mountd.

System Countermeasures: 2 – These traces were obtained from www.incidents.org, so not much is known about what system countermeasures are in place to protect the host from this particular activity. There is one thing to note though in favor of the target system. The attacker sent several requests to the portmapper service over a time period of several seconds, this could indicate that he was not able to elicit a response due to portmapper or mountd not running on the target system.

Network Countermeasures: 1 – After briefly viewing analyzing the tcpdump log from which these detects came, it is apparent that the target network is not well protected. There is a variety of activity detected by Snort which could have easily been blocked by a firewall, such as invalid ip addresses (255.255.255.255) and reflexive TCP ports (port 80).

Severity = (3+1) – (2+1) = 1

9. Defensive recommendations

Unless allowed by the security policy, it is not wise to allow incoming connections to portmapper/mountd. If any stranger was allowed to mount an exportable share from the target system, he could use it to store his own files, or possibly fill up the hard drive on the target system causing some sort of storage DOS. Not to mention there are also various vulnerabilities in mountd (remote command execution, root-account creation). The first course of action I would recommend be to block incoming TCP/UDP connections to the portmapper port either in a router ACL or a firewall. However, if the network mandates that the NFS service be able to external users, there are a few steps one can take to tighten down NFS and limit its usage only to legitimate users.

Most of what can be done to secure and restrict access to NFS can be achieved through modifying the /etc/exports file. To restrict access to an NFS file system by IP address, just specify the IP address after the partition's entry in /etc/exports for example, to restrict access to the /home directory to 192.168.1.101 the entry would look like:

```
/home 192.168.1.101
```

To prevent an NFS client from mounting an NFS file system on the server, one can add the parameter "secure" to an item in /etc/exports:

```
/home 192.168.1.101(secure)
```

To restrict permissions on an NFS file system specify the permissions on that particular item in /etc/exports:

```
/home 192.168.1.101(ro)
```

This list is not all inclusive, there are various other ways to lock down NFS. Common sense also takes precedence too, patches must applied to the system running NFS and to NFS as well.

10. Multiple choice test question

Regarding these traces, the portmapper service which operates over UDP:

a) Like every other network service/daemon on a host, requires that clients connect from a well-known port (in this it would be port 902)
b) Like every other network service/daemon on a host depends on the transport layer's (UDP) functions to send back a response
c) Like every other UDP-based network service/daemon on a host depends on that particular application to respond to clients' requests.
d) Like every other UDP-based network service/daemon on a host, will not respond to clients connecting from a well-known port.

ANSWER: C

DETECT #3

```
07/10-14:21:43.515473  [**] [1:1070:5] WEB-MISC webdav search access [**] [Clas
sification: access to a potentially vulnerable web application] [Priority: 2] {
TCP} 216.127.202.112:3603 -> small.isp.net:80

07/10-14:21:43.515473 216.127.202.112:3603 -> small.isp.net:80
TCP TTL:112 TOS:0x0 ID:9586 IpLen:20 DgmLen:1420 DF
***A**** Seq: 0x11C8464  Ack: 0xD65F53FB  Win: 0x2058  TcpLen: 20
53 45 41 52 43 48 20 2F 41 41 41 41 41 41 41 41  SEARCH /AAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
TRUNCATED FOR LENGTH

07/10-14:21:43.515477  [**] [1:1070:5] WEB-MISC webdav search access [**] [Clas
sification: access to a potentially vulnerable web application] [Priority: 2] {
TCP} 216.127.202.112:3603 -> small.isp.net:80

07/10-14:21:43.515477 216.127.202.112:3603 -> small.isp.net:80
TCP TTL:112 TOS:0x0 ID:9586 IpLen:20 DgmLen:1420 DF
***A**** Seq: 0x11C8464  Ack: 0xD65F53FB  Win: 0x2058  TcpLen: 20
53 45 41 52 43 48 20 2F 41 41 41 41 41 41 41 41  SEARCH /AAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

```
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
TRUNCATED FOR LENGTH
```

1.Source of Trace

These logs were obtained with permission from a former employer (a small ISP and hosting company) whom I had set up an IDS system for. The packet captures are truncated, otherwise they would have taken up much more space.

2. Detect was generated by:

The alerts and subsequent packet dumps were generated by a Snort IDS v2.0.0 running on RedHat 7.2. The alert file was readily available, however the Snort log file was still in binary format and was read into an ASCII text file using the following command:

```
snort –r snort.log > detect.txt
```

The rule which generated this alert is the following and is included in the default ruleset with Snort v2.0.0:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-MISC WebDAV search access"; flow:to_server,established; content: "SEARCH "; depth: 8; nocase;reference:arachnids,474; classtype:web-application-activity; sid:1070; rev:6;)
```

The ISP did have Windows IIS servers on their network but only very few, thus this rule and several others were examined and customized slightly. But core criteria remained essentially the same, this resulted in the following rule:

alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"WEB-MISC WebDAV search access"; flow:to_server,established; content: "SEARCH "; depth: 8; nocase;reference:arachnids,474; classtype:web-application-activity; sid:1070; rev:6;)

The detects were logged by this rule because they matched the 5 main criteria specified by the rule:
1) Communications were occurring using the tcp protocol, with a source IP address specified by the $EXTERNAL_NET variable on any port, going to a destination IP address specified by the $HOME_NET variable on destination port 80

alert tcp $EXTERNAL_NET any -> $HOME_NET 80

2) The payload of the packet contained certain ASCII data specified by the content keyword:

content:"SEARCH";

3) The ASCII content specified is matched, case insensitive

nocase;

4) The maximum search depth is at most 8 bytes from the search point (beginning of payload data):

depth:8;

5) The traffic flow of the packet is towards the $HOME_NET subnet, and the traffic is part of an established session (A+ flags set):

flow:to_server,established;

3.Probability the source address was spoofed:

This particular exploit could be used for either reconnaissance or a denial-of-service attack. In this particular detect it appears that the attacker chose to attempt a denial-of-service attack. Therefore it would be logical for the attacker to spoof the source address, avoiding discovery just in case the target system's administrators had procedures for maintaining an audit trail.

4.Description of the attack:

This particular attack appears to be a buffer overflow targeting Microsoft IIS WebDAV feature, more specifically the SEARCH request feature.

5. Attack Mechanism

The Snort rule that captured this detect suggests that could be one of two activities: recon activity or DoS (http://www.snort.org/snort-db/sid.html?sid=1070). It does not appear that this attack requires any specialized tools or software to use. Apparently, it can be as simple as typing in a URL request in a browser aimed at the target system. First, there would have to be an established web session between the attacker and the target. The basic URL request an attacker would have to make is for "SEARCH", which when processed by vulnerable IIS (running WebDAV) systems could potentially return a directory listing on that server. This URL could look like: `http://www.target.com/SEARCH/`. When the attacker wants to exploit the DoS capability of this vulnerability which is what appears to be happening in this detect, all they would have to do is append an exceptionally long string of repeated characters to the "SEARCH" query which could result in IIS restarting. In this particular detect, the attacker chose to append a long string of the repeated character "A". The reason that this particular vulnerability exists is because there is an unchecked buffer in ntll.dll (a string handling API routine in ntll.dll) and in a Windows component used by WebDAV (for more information: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms03-007.asp). In the actual mechanism of an attack, a request is made to the IIS server. Some of these requests are actually extensions to basic HTTP requests (GET, POST) and include requests such as: PROPFIND, LOCK or SEARCH. This request is processed by the function GetFileAttributesExW, which calls the function RtlDosPathNameToNtPathName_U, which is exported by the ntll.dll. The WebDAV request is not limited in the length of the filename it can request; RtlDosPathNameToNtPathName_U depends on unsigned shorts for string lengths and thus the string cannot be more than 65535 bytes long. The vulnerability depends on this string being at least 65536 bytes long. (http://www.nextgenss.com/papers/ms03-007-ntdll.pdf).

6. Correlations

This particular vulnerability was alerted on by SANS (http://www.sans.org/webcasts/031803.php) which then put out an informative flash broadcast describing the vulnerability, known and theoretical exploits, and defensive recommendations (. ISS's X-force was able to capture several detects which in the webdav request contained the "ccjs". It was also suggested that a character sled of "N"s could be used to perform a buffer overflow attack against WebDAV enabled IIS 5.0 servers. George Guninski (www.guninski.com) also posted code on the Bugtraq for a buffer overflow against the SEARCH method (http://www.securityfocus.com/archive/1/169837).

A search through several prominent websites, mailing lists and forums (including bugtraq, incidents.org, insecure.org, whitehats.com) did not reveal any detects similar to this one. Unfortunately, only the first packet was available in the logs so it is not known whether the attacker tried to append a command at the end of the "A" character sled

which is a common characteristic of buffer overflow attacks. This detect would have to be classified as an attempted DoS.

7. Evidence of active targeting:

This particular detect was directed towards the ISP's network, however the machine it did hit was running a hardened Unix system that wasn't running any http services. Since this particular attack is intended for Windows machines running WebDAV-enabled IIS 5.0, it would have to be classified as a probable "wrong number".

8. Severity:

Severity is calculated using the following formula:

Severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Criticality: 2 – The target system was one of several Unix servers running mail services.

Lethality: 1 – The target system would never be susceptible to this attack, since the attack is intended for Windows systems running a particular application and the target system is a Unix system.

System Countermeasures: 4 – The targeted system was a Unix system with all necessary services shut off and with tcp wrappers wrapped around the necessary applications and services.

Network Countermeasures: 3 – This particular network has an IDS system and a simple packet filtering firewall with very few rules enabled

Severity = (2+1) – (4+3)= -4

9. Defensive Recommendations

This attack hit a Unix system that was invulnerable to the vulnerability. However, there do reside vulnerable Windows IIS 5.0 systems on the network, and several steps can be taken to protect against this attack as suggested by SANS:

- Apply the Microsoft patch (MS03-07) to vulnerable systems
- Disable WebDAV
- Restrict the URL buffer size to recommended size (16kb)

10. Multiple choice test question:

```
07/10-14:21:43.515477 216.127.202.112:3603 -> small.isp.net:80
TCP TTL:112 TOS:0x0 ID:9586 IpLen:20 DgmLen:1420 DF
***A**** Seq: 0x11C8464  Ack: 0xD65F53FB  Win: 0x2058  TcpLen: 20
53 45 41 52 43 48 20 2F 41 41 41 41 41 41 41 41  SEARCH /AAAAAAAA
```

```
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

Which of the following is most likely in the trace above?

a) A scan for IIS 5.0 webservers
b) A request for a directory listing
c) A buffer overflow attempt using non-standard http request
d) A buffer overflow attempt using a standard http request

ANSWER: C

ASSIGNMENT #3

Executive Summary

A security audit was provided at the request of the University based on several days' worth of alert, scans, and oos logs. Several custom Java programs were written to process through these logs and mine relevant information that would be useful in the analysis. This data was analyzed and the analysis divided into two categories: Volume Analysis and Severity Analysis, of which several interesting patterns and detects were discovered. However, the analysis was only of limited value due to the fact that only Snort alert logs, Scans logs, and OOS logs were provided. Additional verification of actual attacks would have occurred and would have been much more effective had the University provided information about the network topology and had provided payload data, which would have greatly aided in the confirmation of suspect alerts. Nonetheless, defensive recommendations were made as best as possible given the data.

Files Analyzed

The University provided five consecutive days' worth of log files to be analyzed: April 18, 2003 through April 22, 2003, it should be noted however that for the OOS logs, the date suggested by log file name did not correspond to the timestamp and dates as reported IN the log file. For example, the name suggested by log file OOS_Report_2003_04_19_8227 implied that the data contained within was for the 19th of April, however, only at the end of the log file would you find a very small amount of data for the 19th. The rest of the file (over 90%) contained data for the 18th as suggested by timestamps. This was the case for the rest of the OOS logs, therefore the data suggested by the log file name actually contains data for the previous day. That said, the log files were analyzed were:

ALERT	PORTSCAN	OOS
alert.030418	scans.030418	OOS_Report_2003_04_19_8227
alert.030419	scans.030419	OOS_Report_2003_04_20_8227
alert.030420	scans.030420	OOS_Report_2003_04_21_8227
alert.030421	scans.030421	OOS_Report_2003_04_22_8227
alert.030422	scans.030422	OOS_Report_2003_04_23_8227

It is recommended that the next time files are provided by the University that packet dumps are included as well so that a more complete analysis can occur. The reason for this request is because for the files provided, especially the alert files, the only way to confirm whether an alert was triggered by an actual attack or is a false positive is to view the packet payload. Otherwise, there is no way tell whether an attack actually occurred and the analyst must make some assumptions that he would not have normally.

Analysis Description

The first step in the analysis process involved the development of custom Java programs that could parse through each of the different types of logs (alerts, scans, and OOS logs) provided by the university. In all, 3 java programs were developed: Scans.java, Alert.java, and OOS.java. The purpose of these programs was to compute various statistics such as total counts and unique counts, and also to find any relationships among the various IP addresses that generated these logs. However several issues had to be addressed when coding since there were several problems with the logs. One of the first issues addressed in writing these programs was log corruption, some of the logs appeared to be corrupted, with missing destination IPs, and incomplete log entries. Each of these programs was customized to handle these inconsistencies by ignoring corrupted entries and not including them in the final statistics. The next issue addressed was inconsistent log format, particularly for the snort generated Alerts; more specifically, some alert entries would not be followed by source and destination IP addresses, particularly "spp_portscan" alerts; this was accounted for by including those specific alerts in the statistics for the number of alerts but not including them in statistics for source and destination IP addresses.

After these issues were addressed and coded for the next phase entailed organizing the provided files for analysis by the programs. As stated before, 5 days worth of logs was provided by the University for analysis; these were concatenated together according to whether they were Alerts, Scans or OOS logs.

1. To generate the file to be analyzed by Alerts.java the alerts files were concatenated together by using the command:

```
cat alert.030418 alert.030419 alert.030420 alert.030421 alert.030422
> alertapr18thru22.txt
```

2. To generate the file to be analyzed by Scans.java the scans files were concatenated together by using the command:

```
cat scans.030418 scans.030419 scans.030420 scans.030421 scans.030422
> scansapr18thru22.txt
```

3. To generate the file to be analyzed by oos.java the oos files were concatenated together by using the command:

```
Cat OOS_Report_2003_04_19_8227 OOS_Report_2003_04_20_16512
OOS_Report_2003_04_21_32071 OOS_Report_2003_04_22_9834
OOS_Report_2003_04_23_30637 oosapr18thru22.txt
```

Each of the resulting files were then analyzed by their appropriate programs to generate statistics:

1. `java Alert alertapr18thru22.txt`

2. `java Scans scansapr18thru22.txt`
3. `java Scans oosapr18thru22.txt`

The resulting output can be found later in the Analysis section of this report and source code for all the programs can be found in the Reference section.

Once all pertinent statistics were generated, focus shifted to analysis. Analysis was performed on the concatenated files as a whole except in a few specified instances. The output was then analyzed and several key pieces of information were then extracted from these statistics. These include from across the board: the most prevalent source and destination IP addresses, the most prevalent alerts, the most accessed well-known ports (external and internal), the most accessed ephemeral ports (external and internal), and other custom ones according to log file type.

The next step consisted of constructing a volume and severity matrices; these were based on which IP addresses were conducting scans, triggering alerts, and producing OOS packets. The matrices were used to determine: the volume of attacks/scans/targets and which IP addresses, especially external, were conducting the most complex attacks. Complex meaning how many and the variety of alerts triggered; the amount of scans and the ports targeted, if there are scans as well as alerts, and a variety of other factors. The purpose is to discover how many and if any concerted efforts are being directed toward the University's network. Emphasis was not placed on the number of times a particular alert occurred but on what other alerts it alerted with. For example, if there are 100 instances of a particular alert towards a host does it mean that a hacker tried the same attack 100 times? The odds are that they are the result of false positives stemming from normal traffic and poor signatures or misconfigured devices. Activity that would probably be indicative of a genuine attacker, would include varied attacks and activity against a single IP, not the same attack 100 times in a row (unless it is an attacker scanning a range of addresses). The analysis was based on that conjecture, thus one of the goals was to find correlation among the different types of logs. For example, did the attacker scan the target host to discover which services were available (detected in the scans logs), then try sending Out-of-spec packets to try to discover host OS (detected in the OOS logs), and then finally try specific attacks based on his findings (detected in the alert logs). In constructing the matrix, the most complex attacks were hand-picked from the statistics generated from the custom java programs. Although the complexity can perhaps be viewed as subjective, it is the analyst's personal opinion that severity based on complexity is better than severity based on the volume of attacks.

Volume analysis

A top talkers list was generated based on each and/or a combination of the log files provided. There were several objectives in doing this. The first objective was to discover the most active attackers. The second objective was to discover which hosts on the University's network were most actively targeted by scans. The third objective was to determine what were the most popular targeted ports.

To achieve the first objective, the top 10 attackers were determined by counting their number of log entries in the concatenated scans log file.

Scans – Top 10 Active External Source IPs	Count
1. 146.164.34.42	12962
2. 193.11.250.21	11323
3. 213.84.229.115	10926
4. 217.40.73.165	10374
5. 217.70.4.246	9202
6. 216.137.3.107	8838
7. 81.56.209.187	7212
8. 152.1.193.6	6750
9. 158.36.40.5	4122
10. 80.14.15.28	3010

The top 10 scanning external source IPs were listed so that the University may identify and block the most active IPs/subnets that had no legitimate reason for their scanning. It is interesting to note that volume of scans did not correlate with period of time over which the scanning occurred; overall scanning occurred anywhere from a range of 30 minutes to a few hours. The top two IPs 146.164.34.42 and 193.11.250.21 each scanned for a very short period of time, approximately 30 minutes, and both scanned a large portion of the University's class B subnet for TCP port 443. Of particular note, 213.84.229.115 scanned for 13 hours straight beginning at approximately 1500hours on April 19th ending approximately 0400hours on April 20th. Apparently, the only IP being scanned was 130.85.195.163 which was the top targeted internal destination IP. Scanning activity for this pair included many packets with odd or invalid TCP flag combinations (esp. null scans – TCP packets with no TCP flags set) directed towards 130.85.195.163, but oddly enough this did not show up in the OOS logs. Here is a snippet of that traffic:

```
Apr 19 15:01:42 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:01:47 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:01:52 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:01:55 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:01:55 213.84.229.115:59479 -> 130.85.195.163:2829 INVALIDACK
*2*A*R*F
Apr 19 15:01:58 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:02:01 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:02:03 213.84.229.115:58680 -> 130.85.195.163:41436 UNKNOWN
12UA**** R
Apr 19 15:02:04 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:02:12 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:02:14 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:02:20 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:02:22 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:02:24 213.84.229.115:34946 -> 130.85.195.163:47254 SYNFIN
1*****SF RE
Apr 19 15:02:31 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:02:46 213.84.229.115:0 -> 130.85.195.163:0 NULL ********
Apr 19 15:02:46 213.84.229.115:46267 -> 130.85.195.163:58600 FULLXMAS
*2UAPRSF
```

The fact that this IP was able to scan for so many hours is possibly indicative of a poor security policy, poor perimeter security, or no human intervention, this was noted in the defensive recommendations section. The other IPs that recorded short targeted scans were

also recommended as possible candidates to be blocked by a perimeter device, these are: 217.40.73.165, 81.56.209.187, 152.1.193.6, 158.36.40.5, and 80.14.15.28. It should be noted that though unlikely, it is possible that some IPs in this list may not actually be scanning and the activity detected could be a part of normal network activity.

To achieve the second objective, the top 10 internal scan targets were determined by counting their number of log entries in the concatenated scans log file.

Scans – Top 10 Targeted Internal Destination IPs	Count
130.85.195.163	10928
130.85.203.230	6752
130.85.208.222	428
130.85.249.194	417
130.85.87.244	325
130.85.194.223	289
130.85.236.146	200
130.85.220.178	197
130.85.24.22	190
130.85.196.171	128

The top 10 targeted internal destination IPs were listed so that the University may identify those internal hosts which are the most targeted by outside attackers and take steps to mitigate the risk against those hosts. There are several interesting things to note. 130.85.195.163 and 130.85.203.230 appear to be much more targeted hosts than others on the list, with counts in the thousands as opposed to the other IPs which had counts in the hundreds. 130.85.195.163 was the target of extended https scans with odd TCP flags combinations. The number two targeted internal IP 130.85.203.230 was the target of SYN scans for relatively high ephemeral ports, possibly indicative of a scan for a trojaned host. Grepping through the alert or oos files did not reveal any evidence of returned responses, but that does not mean there are no hosts infected with a virus or trojaned. However, that could also depend on the type of signatures that are deployed on sensors, especially one that look for signs of infected or trojaned hosts. There are 3389 unique internal destination IPs in the concatenated scans log which means that IPs that are NOT in the top 10 list have only been scanned at most several hundred times. This could indicate that the top 10 hosts: are some sort of high-volume servers and that the scanning activity is legitimate(consultation with the University's systems administrators are necessary to confirm this), are favorite targets of hackers, are relatively unprotected when compared to other hosts, and/or are the target of a DOS attack. It should be noted that some IPs in this list may not be scan targets are but the scanning activity detected could be a part of normal network activity.

To achieve the first part of the third objective, the top 10 targeted well-known ports were determined by counting the number of log entries in the concatenated scans log file.

Scans – Top 10 targeted Internal Well-Known ports	Count
443	25432
445	22417

139	16782
80	13892
0	8791
21	5795
135	3541
25	1468
22	164
110	93

The top 10 targeted well-known ports were listed to get an idea of the type of activity present on the network (many "scans" could actually be due to legitimate activity from chatty protocols), to discover if any particular services were being targeted, to discover if there were any strange ports being accessed, and to get an idea of the security policy as applied to network perimeter security devices. There does not appear to be anything terribly unusual about which ports were being scanned. There were 180 unique internal destination well-known ports scanned. The number one port being scanned was port 443 which carries https traffic over TCP. Scans for port 443 were short and targeted, occurring all in one day, April 20th. Other target ports include 135,139, and 445 which are used for Windows NetBIOS; 80 which is used to carry http traffic; 0, which is commonly associated with fragmented traffic; 21, which is the command port for ftp; 22, which carries ssh; 25, which carries smtp traffic; and 110 which is used for mail retrieval using the POP3 protocol. Consultation with the University's systems administrators are necessary to determine if any of these ports should be open to the public, though they seemed normal because all carried traffic associated with normally used web-applications.

To achieve the second part of the third objective, The top 10 targeted ephemeral ports were determined by counting the number of log entries in the concatenated scans log file.

Scans – Top 10 targeted Internal Ephemeral ports	Count
1433	23277
4000	2603
1080	1794
8080	1785
3128	1773
6588	1762
4588	1744
3389	1562
6346	1258
6112	942

The top 10 targeted ephemeral ports were listed to discover any potential Trojan/backdoor ports, as many use high-numbered ephemeral ports. Several of the ports scanned are worthy of note. Port 1433 is the port for Microsoft's SQLserver which was the target of the SQLslammer worm earlier in the year. While grepping through the alert file for alerts on port 1433, there did not appear to be any attacks aimed specifically at the

MS-SQLserver application. However, that could be due to there were no signatures employed to detect SQL specific attacks. This was the only scan which was in the tens of thousands. Second in volume was port 4000 used by Terabase, which is a search engine for highly complex databases; this activity could be explained by the fact that many universities house large databases, especially in departments which are involved in any type of research whose data either needs to be publicly accessible or exchanged with other institutions, of course consultation with the systems administrators is needed to confirm this conjecture. Other services to note (as implied by their port number) include: http-proxy, squid-proxy, socks-proxy, gnutella, dtscpd, and ms-wbt-server. Many scans in the log seemed like they could have easily been blocked at a router (using ACLs) or simple packet filtering firewall. Many of the ports that were able to get through it seems, should not be allowed by a "Deny all not specifically allowed" security policy (proxy ports, Gnutella). This observation is noted in the "Defensive Recommendations" section.

Severity Analysis

A different approach was used when analyzing the alerts files; in addition to counting the volume of alerts (counting can only tell so much as sometimes all it takes is one packet to successfully compromise a host), IPs were selected by the variety and seriousness (results in compromise, or symptoms of compromise) of alerts they triggered on, this approach was taken in the hopes of finding the most serious attackers who were willing to try more than one method of attack and had a strategy of attack. Another goal was to find internal hosts that were possibly infected by a virus, trojaned, or communicating via covert channel. They were hand picked by manually parsing through the output generated by the Alerts.java program on the concatenated Alerts file. The three categories that were of most concern were Internal Source IPs, Internal Destination IPs, and External Source IPs. An Alert Analysis was then performed as part of the Severity Analysis to help find authentic attacks. An attempt to find correlation with the other types of log files was also incorporated to help find authentic attacks.

Alerts for Internal Source IPs		Count
MY.NET.97.88	• IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	40
	• NIMDA - Attempt to execute cmd from campus host	7
MY.NET.97.191	• NIMDA - Attempt to execute cmd from campus host	2
	• IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	3
	• spp_http_decode: IIS Unicode attack detected	4
MY.NET.251.70	• spp_http_decode: IIS Unicode attack detected	19
	• spp_http_decode: CGI Null Byte attack detected	1
	• High port 65535 udp - possible Red Worm - traffic	3
	• TFTP - Internal TCP connection to external tftp server	1556

MY.NET.98.35	• IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	4
	• NIMDA - Attempt to execute cmd from campus host	1
MY.NET.201.106	• spp_http_decode: CGI Null Byte attack detected	2
	• spp_http_decode: IIS Unicode attack detected	14
	• TFTP - Internal TCP connection to external tftp server	11
	• High port 65535 tcp - possible Red Worm – traffic	12
	• High port 65535 udp - possible Red Worm – traffic	4

Alerts for External Source IPs		Count
213.84.229.115	• Null scan!	9229
	• Probable NMAP fingerprint attempt	116
	• SYN-FIN scan!	21
	• Queso fingerprint	15
	• High port 65535 tcp - possible Red Worm – traffic	8
219.52.154.110	• Null scan!	294
	• Queso fingerprint	2
	• High port 65535 tcp - possible Red Worm – traffic	2
	• SYN-FIN scan!	1
24.159.126.37	• External RPC call	1
	• connect to 515 from outside	1
	• Possible trojan server activity	16
131.118.254.130	• EXPLOIT x86 setgid 0	3
	• EXPLOIT x86 NOOP	68
	• EXPLOIT x86 setuid 0	2
	• EXPLOIT x86 stealth noop	11
66.196.72.55	• CS WEBSERVER - external web traffic	25
	• MY.NET.30.4 activity	

Alerts for Internal Destination IPs		Count
MY.NET.195.163	• Null scan!	9224
	• Probable NMAP fingerprint attempt	116
	• SYN-FIN scan!	21
	• Queso fingerprint	15
	• High port 65535 tcp - possible Red Worm - traffic	6

	• SMB Name Wildcard	4
MY.NET.24.47	• SMB Name Wildcard	10
	• FTP passwd attempt	7
	• Queso fingerprint	2
	• FTP DoS ftpd globbing	1
MY.NET.24.23	• Watchlist 000220 IL-ISDNNET-990517	2
	• Queso fingerprint	100
	• SMB Name Wildcard	72
	• High port 65535 tcp - possible Red Worm - traffic	2
	• Bugbear@MM virus in SMTP	1
MY.NET.205.118	• EXPLOIT x86 setgid 0	1
	• SMB Name Wildcard	18
	• High port 65535 udp - possible Red Worm - traffic	2
	• EXPLOIT x86 setuid 0	3
	• [UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	1
	• EXPLOIT identd overflow	12156
MY.NET.225.66	• SMB Name Wildcard	236
	• spp_http_decode: CGI Null Byte attack detected	61
	• spp_http_decode: IIS Unicode attack detected	323
	• Watchlist 000220 IL-ISDNNET-990517	227
	• Queso fingerprint	22
	• TFTP - Internal TCP connection to external tftp server	4
	• Possible trojan server activity	8
	• NMAP TCP ping!	1
	• High port 65535 tcp - possible Red Worm - traffic	2
	• EXPLOIT x86 setgid 0	1
	• EXPLOIT x86 setuid 0	1

Alert Analysis

Alert analysis was based on selecting and analyzing those alerts from the Severity Analysis which were alerted on. In the first part of Alerts Analysis, the Internal Source

IPs section was singled out to help identify those alerts which indicate possibly infected hosts or otherwise illegimate activity. An attempt was made to place these alerts into those different classes where applicable and to qualify those alerts as either "Authentic" or "False Positive" depending on a variety of factors especially on how difficult it is to trigger those alerts. The second part of Alerts Analysis involved analyzing external source IPs which appeared to have the most coordinated and planned attacks.

The latest version of Snort that was available during the timestamp of the log files was version 2.0.0. Thus it is assumed that the ruleset used to generate alerts in the alerts log files was from that version of snort.

The unique alerts which alerted for Internal Source IPs and were of concern were:
NIMDA - Attempt to execute cmd from campus host
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize
spp_http_decode: IIS Unicode attack detected
spp_http_decode: CGI Null Byte attack detected
High port 65535 udp - possible Red Worm – traffic
TFTP - Internal TCP connection to external tftp server

NIMDA - Attempt to execute cmd from campus host

This alert is a custom alert, thus there is no official documentation and one can only infer from the name the specific signature details employed to detect this attack "from campus host". There are several things we can glean from the alert file, the logs, and from the name of the alert. First, the source host specified before the "->" at the head of the snort rule should be from the MY.NET subnet which satisfies the "from campus host" criteria, this is also evident in the alert file logs as all the source IPs in the log for this signature are from the MY.NET subnet (4 unique hosts alerted, MY.NET.97.191, MY.NET.249.214, MY.NET.98.35, MY.NET.97.88). This alert seems to trigger only on destination port 80 as seen in the alert log files:

```
04/18-22:16:39.599407  [**] NIMDA - Attempt to execute cmd from campus
host [**]
 MY.NET.97.191:3709 -> 64.122.72.107:80
04/18-22:48:45.699087  [**] NIMDA - Attempt to execute cmd from campus
host [**]
 MY.NET.249.214:4397 -> 64.33.51.156:80
04/18-22:28:17.566855  [**] NIMDA - Attempt to execute cmd from campus
host [**]
 MY.NET.97.191:4743 -> 169.132.74.100:80
04/22-02:25:19.758403  [**] NIMDA - Attempt to execute cmd from campus
host [**]
 MY.NET.98.35:2200 -> 130.49.66.95:80
04/22-03:35:10.223836  [**] NIMDA - Attempt to execute cmd from campus
host [**]
 MY.NET.97.88:2649 -> 130.158.117.81:80
04/22-03:35:47.660369  [**] NIMDA - Attempt to execute cmd from campus
host [**]
 MY.NET.97.88:4029 -> 130.158.200.14:80
04/22-03:39:22.758274  [**] NIMDA - Attempt to execute cmd from campus
host [**]
```

```
 MY.NET.97.88:2160 -> 207.7.10.226:80
04/22-03:53:30.933006  [**] NIMDA - Attempt to execute cmd from campus
host [**]
 MY.NET.97.88:2662 -> 130.158.55.178:80
04/22-03:54:15.717222  [**] NIMDA - Attempt to execute cmd from campus
host [**]
 MY.NET.97.88:3936 -> 130.158.213.94:80
04/22-03:54:18.007722  [**] NIMDA - Attempt to execute cmd from campus
host [**]
 MY.NET.97.88:2573 -> 209.229.121.151:80
04/22-03:29:39.649122  [**] NIMDA - Attempt to execute cmd from campus
host [**]
 MY.NET.97.88:1068 -> 130.150.183.119:80
```

However, all the rules present for Nimda that come with Snort-2.0.0 are written to trigger on destination port 139. We have to assume that the Nimda attack vector this alert is attempting to detect is the Unicode Web Traversal Exploit which targets Microsoft Web Server software IIS 4.0 and 5.0 (http://www.microsoft.com/technet/security/bulletin/ms00-078.asp) This exploit entails constructing a specially crafted URL to perform a directory traversal to gain access to cmd.exe executable, which when passed the proper parameters can be used to execute a variety of commands on the target system with the privilege of the IUSR_machinename account. In the actual payload of the packet, the string "cmd.exe" or "root.exe" would have to be present (http://www.cert.org/advisories/CA-2001-26.html). A snort rule based on these conjectures would probably resemble:

```
alert tcp $HOME_NET any -> any 80 (msg: " NIMDA - Attempt to execute
cmd from campus host"; content:"cmd.exe")
```

There aren't a lot of reasons why the string "cmd.exe" should be present in any payload destined for port 80, although it can false positive say for example if someone is doing research on the web on Nimda and happens to download a description which contains the string "cmd.exe", this has happened in the analyst's personal experience several times. However, this alert is difficult to trigger by chance, so it is being classified as "Authentic". The chances that the alerts represent an actual NIMDA infection however is questionable, since NIMDA tries many different URL variants in its scanning phase; these particular IPs are only alerting no more than a few times on NIMDA. It is recommended that any internal source IP which triggers this alert be investigated immediately and be disconnected from the network.

IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize

This signature is one available at whitehats.com (http://www.whitehats.com/info/IDS552). This signature is meant to detect attacks attempting to exploit a buffer overflow vulnerability in Microsoft's IIS software. More specifically, overflowing an unchecked buffer in the Index Server ISAPI extension could result in the IIS server's compromise. (http://www.microsoft.com/technet/security/bulletin/MS01-033.asp). This vulnerability

exists because there is a section of code in idq.dll that has an unchecked buffer in it. The ISAPI extension which handles URL input (for example, default.ida) can be exploited to gain full privileges under the SYSTEM account. All an attacker would have to do is send a specially crafted URL consistent with a buffer overflow in an established web session to a vulnerable IIS server. It might look something like this (from http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids552&view=signatures):

```
06/19-12:53:01.728385 10.51.141.239:1505 -> 10.10.00.01:80
TCP TTL:61 TOS:0x0 ID:59632 IpLen:20 DgmLen:350 DF
***AP*** Seq: 0xEB4F0CDC  Ack: 0x68644817  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 62082230 0
47 45 54 20 2F 4E 55 4C 4C 2E 69 64 61 3F 58 58  GET /NULL.ida?XX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 3D 58 20 48 54 54 50 2F  XXXXXXXX=X HTTP/
31 2E 31 0D 0A 48 6F 73 74 3A 20 32 34 2E 32 37  1.1..Host: 10.10
2E 36 38 2E 38 33 0D 0A 0D 0A                    .00.01....
```

The Snort signature meant to capture an attempted exploit as specified by WhiteHats is:

```
alert TCP $EXTERNAL any -> $INTERNAL 80 (msg: "IDS552/web-iis_IIS ISAPI
Overflow ida";dsize: >239; flags:A+;uricontent: ".ida?";classtype:
system-or-info-attempt;reference:arachnids,552;)
```

This signature is prone to false positives and is not particularly effective for several reasons. Doug Kite (GCIA #0609) also reported a multitude of this activity except that he detected alerts in the thousands as opposed to the dozens reported this particular audit. We must analyze the main triggers of this alert to understand this. The purpose of this signature is to detect a buffer overflow in an established web session against the ISAPI extensions (*.ida); the "flags:A+", and "uricontent: .ida?" components of the signature do well to detect established requests for the ISAPI extension *.ida. But, the "dsize:>239" component is not sufficient to detect a buffer overflow. Any URL request which results in the packet payload size being larger than 239bytes (with other criteria satisfied as well, of course) is prone to trigger this signature. It is not uncommon to find very long URLs, because of that and because there are over 1000 such alerts, any host which triggers this alarm only are classified as "False Positive". There are several interesting things to note concerning this alarm. For only Internal Source IPs (3 unique hosts alerted, MY.NET.97.191, MY.NET.98.35, and MY.NET.97.88) this alert only showed up when

paired with the alert "NIMDA - Attempt to execute cmd from campus host". Without actual payload data and logs, one can only surmise as to why this is the case. The strategy here would be to combine the key triggers of both signatures to infer what happened. From the first inferred signature, for "NIMDA…..", it is assumed that the "cmd.exe" string can be anywhere within the payload of the packet. From the second known signature, we know that the URL must result in there being at least a 239byte size packet that will hopefully catch a buffer overflow. The purpose of most buffer overflows is to take control of the execution path that a particular buffer is in, and instead replace it with an attacker's commands/code. What could be happening is that a buffer overflow was attempted against the target host(s) with "cmd.exe" appended to the end of that overflow with the hopes of executing. Therefore, all hosts which alert on ISAPI overflow AND NIMDA are classified as "Authentic". Once again, payload data would be needed to confirm anything at all.

spp_http_decode: IIS Unicode attack detected & spp_http_decode: CGI Null Byte attack detected

Both of these alerts are part of the http_decode preprocessor. Basically, after decoding of packets and before they are passed through the detection engine, http_decode will normalize http requests from their encoded form into their ASCII equivalents. When the double_encode option is specified in the snort.conf file for http_decode.,"IIS Unicode attack detect" will alert on URL requests where the percent sign has been encoded into hexadecimal, double encoded so to speak. For example, in the actual URL request you might see "%255c" where the "%25" translates into "%" and after a second round of decoding %255c would translate into "/". It is not uncommon to find Unicode encoded text over http, so there is a good possibility that this alert will false positive. For the "CGI Null Byte attack", if the http_decode preprocessor decodes and finds a "%00" in a URL request it will alert. However this alert is prone to giving false positives on SSL encrypted traffic and cookies with urlencoded binary data. Both alerts are also subject to false positive on malformed http requests and are infamous on security mailing lists to generating many false positives. All IPs (MY.NET.251.70, MY.NET.201.106) that triggered on these alerts also alerted on "TFTP - Internal TCP connection to external tftp server" and "High port 65535 udp - possible Red Worm – traffic", judgement about whether or not the IPs that generated these alerts are "Authentic" will be held until the next section when those alerts are also analyzed.

TFTP - Internal TCP connection to external tftp server

This alert is a custom alert, as with the NIMDA alert, several pieces of information can be gleaned from the name of this alert and the rule that triggers it can be partially inferred. First, this alert looks for internal hosts using TCP as the transport protocol:

```
alert TCP $HOME_NET any ->
```

The second part of the rule (to completion) looks for a connection (it is not known if this is for an initial or established connection) to an external host for the TFTP service (port 69):

```
alert TCP $HOME_NET any -> $EXTERNAL_NET 69 (msg:"TFTP – Internal TCP
connection to external tftp server")
```

The name of the rule does not hint at sort of content-based triggering, but we cannot know for sure. TFTP, which stands for Trivial FTP, is basically a very simple ftp service which runs over udp and does not provide for much in the way of security. It is often used to boot diskless workstations and back up (Cisco) router configuration files, and IOS images . TFTP can be a security problem because the service is not password protected, and systems can be left wide open if the service is not shut off. This (inferred) rule is easy to trigger and is thus being classified as "Authentic".

High port 65535 udp - possible Red Worm – traffic

This alert was also a custom alert as well, however it is harder to deduce the rule from name of the alert than it is for the other alerts. There are no keywords such as "from", "to", "external", or "internal" to help infer the direction of traffic in which this rule is alerting on. The only keywords which give any hint as to how the rule is written for this alert are the port number and protocol type, 65535 and udp respectively. A bit of research (http://www.giac.org/practical/gsec/Anthony_Dell_GSEC.pdf) revealed more about the mechanism of the Red Worm and how a rule might be written to detect resultant traffic. The Red worm also known as the Adore worm, tries exploits for several services including LPRng, rpc-statd, and BIND. It also replaces several binaries with Trojan versions, of particular importance is klogd which is replaced with a program named "icmp". When icmp receives a particular ICMP packet which is 77 bytes in length, it starts a backdoor on TCP (or as suggested by this alert, UDP) port 65535 which gives root access to anyone who telnets to that port; it also sends captured system information to several email addresses. From this, we can deduce that the header of the Snort alert rule might look something like:

For established backdoor sessions

```
alert UDP $HOME_NET 65535 -> $EXTERNAL_NET any(flags:A+;)
```

or for connecting to the backdoor:

```
alert UDP $EXTERNAL_NET any -> $HOME_NET 65535(flags:S;)
```

The content portion of the rule (if there is one) would have to be considerably dynamic, since a telnet session is available over port 65535. Triggering this alert would not exactly be difficult but a bit more uncommonplace for a reason. Port 65535 is the last valid port in the range of ephemeral ports available, for someone to initiate a connection to the backdoor would require a ephemeral port to port 65535 connection which is unusual since most services are bound to well-known (<1024) ports. It is also a bit unusual,

though perfectly legitimate, to connect from port 65535 to a well-known service, this would definitely generate alerts on the first possible rule for established backdoor. sessions. In the concatenated alerts file, there were 35456 alerts for Red Worm (tcp and udp) and 10322 alerts for Red Worm udp alone. All in all Red Worm traffic accounted for approximately 10% of total traffic (35456 / 334854 total alerts), this means either there are a lot of Red Worm infected hosts or the Snort rule is too loosely written and is generating a lot of false positives. On its own accord, this alert would have to be classified as "False Positive", based on the inferred rule and volume of alerts.

Referring back to whether or not the http_decode alerts (spp_http_decode: IIS Unicode attack detected & spp_http_decode: CGI Null Byte attack detected) are "Authentic", we must try to comprehend why these alerts occurred collectively with the TFTP and Red Worm alerts on MY.NET.251.70 and MY.NET.201.106. First, three out of the four alerts trigger on port 80 and one alert on port 69. These could possibly be attributed to Unicode-encoded URL requests that originate from the MY.NET clients on port 65535 with some data being transferred over UDP. The TFTP alert is more unusual. As stated before, TFTP is often used to boot diskless workstations and for backing up certain router information. So this brings about two possibilities, the hosts that triggered these alerts are either routers or diskless workstations; being that the systems analyzed are in a University environment, it is common to find diskless workstations. If these hosts were routers, one would expect router backups (possibly through TFTP) on a regular basis; however grepping through the logs reveals that the TFTP alerts for both IPs occur in a relatively short amount of time, less than a day for MY.NET.251.70 (truncated):

```
04/21-04:17:43.375127  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] 81.5.166.85:69 -> MY.NET.251.70:4840
04/21-04:17:47.086533  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.251.70:4840 -> 81.5.166.85:69
04/21-04:17:48.383176  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.251.70:4840 -> 81.5.166.85:69
04/21-04:07:27.360767  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] 81.5.166.85:69 -> MY.NET.251.70:4782
04/21-04:17:49.129661  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.251.70:4840 -> 81.5.166.85:69
04/21-04:17:50.339929  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.251.70:4840 -> 81.5.166.85:69
04/21-04:17:59.055758  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.251.70:4840 -> 81.5.166.85:69
04/21-04:07:36.359178  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.251.70:4782 -> 81.5.166.85:69
04/21-04:18:02.194035  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.251.70:4840 -> 81.5.166.85:69
04/21-04:18:14.187863  [**] TFTP - Internal TCP connection to external
tftp serv
```

```
er [**] MY.NET.251.70:4840 -> 81.5.166.85:69
04/21-04:18:14.196894  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] 81.5.166.85:69 -> MY.NET.251.70:4840
04/21-04:18:14.198092  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.251.70:4840 -> 81.5.166.85:69
04/21-04:18:14.198221  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.251.70:4840 -> 81.5.166.85:69
04/21-04:07:54.680745  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.251.70:4782 -> 81.5.166.85:69
```

and in the case of MY.NET.201.106, less than a minute!!:

```
04/18-08:49:14.911451  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] 63.231.14.237:69 -> MY.NET.201.106:4409
04/18-08:49:15.945433  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.201.106:4409 -> 63.231.14.237:69
04/18-08:49:17.485979  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] 63.231.14.237:69 -> MY.NET.201.106:4409
04/18-08:49:17.804963  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.201.106:4409 -> 63.231.14.237:69
04/18-08:49:18.697506  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] 63.231.14.237:69 -> MY.NET.201.106:4409
04/18-08:49:20.488588  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.201.106:4409 -> 63.231.14.237:69
04/18-08:49:21.076290  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] 63.231.14.237:69 -> MY.NET.201.106:4409
04/18-08:49:21.078798  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.201.106:4409 -> 63.231.14.237:69
04/18-08:50:10.871430  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.201.106:4409 -> 63.231.14.237:69
04/18-08:50:43.508948  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.201.106:4409 -> 63.231.14.237:69
04/18-08:50:43.510220  [**] TFTP - Internal TCP connection to external
tftp serv
er [**] MY.NET.201.106:4409 -> 63.231.14.237:69
```

The chances are that MY.NET. 201.106 and MY.NET.251.70 are diskless workstations and not routers. Notice that both hosts are apparently attempting repeated connection attempts to the external tftp servers with responses back from the external IPs, MY.NET.251.70 to 81.5.166.85, and MY.NET.201.106 to 63.231.14.237. One would think that if these were routers that they would back up their files to host on a local or

possibly neighboring network. Instead they are making connections on a relatively uncommon used service to addresses way outside their network. For instance, 81.5.166.85 resolves to a network in Great Britain, and 63.231.14.237 resolves to a network in Colorado, TFTP connections across such distances seem a little odd and suspicious. It is possible that 251.70 and 201.106 are misconfigured diskless workstations. Regarding the alerts that trigger on port 80 collectively with the suspicious tftp alerts/activity, they are classified as "Authentic".

For the second part of Alerts Analysis, external hosts which appeared to have the most coordinated and strategic plan of attack were identified and analyzed. The hosts that were identified were 213.84.229.115 and 131.118.254.130.

213.84.229.115 alerted on the following: "Null scan!", "Probable NMAP fingerprint attempt, SYN-FIN scan!", "Queso fingerprint", "High port 65535 tcp – possible Red Worm – traffic". 219.52.154.110 triggered on the same alerts except for "Probable NMAP fingerprint attempt", this is why 213.84.229.115 was chosen as having a more coordinated attack. The rule for "Probable NMAP fingerprint attempt" is hard to false positive on; searching on this exact alert message did not bring up any Snort alert documentation however, searching for the keywords "NMAP fingerprint attempt" in the Snort alert documentation did bring up an alert documentation for "SCAN nmap fingerprint attempt". The rule for this particular alert is:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap
fingerprint
attempt";flags:SFPU;reference:arachnids,05;classtype:attempted-
recon;sid:629;rev:1;)
```

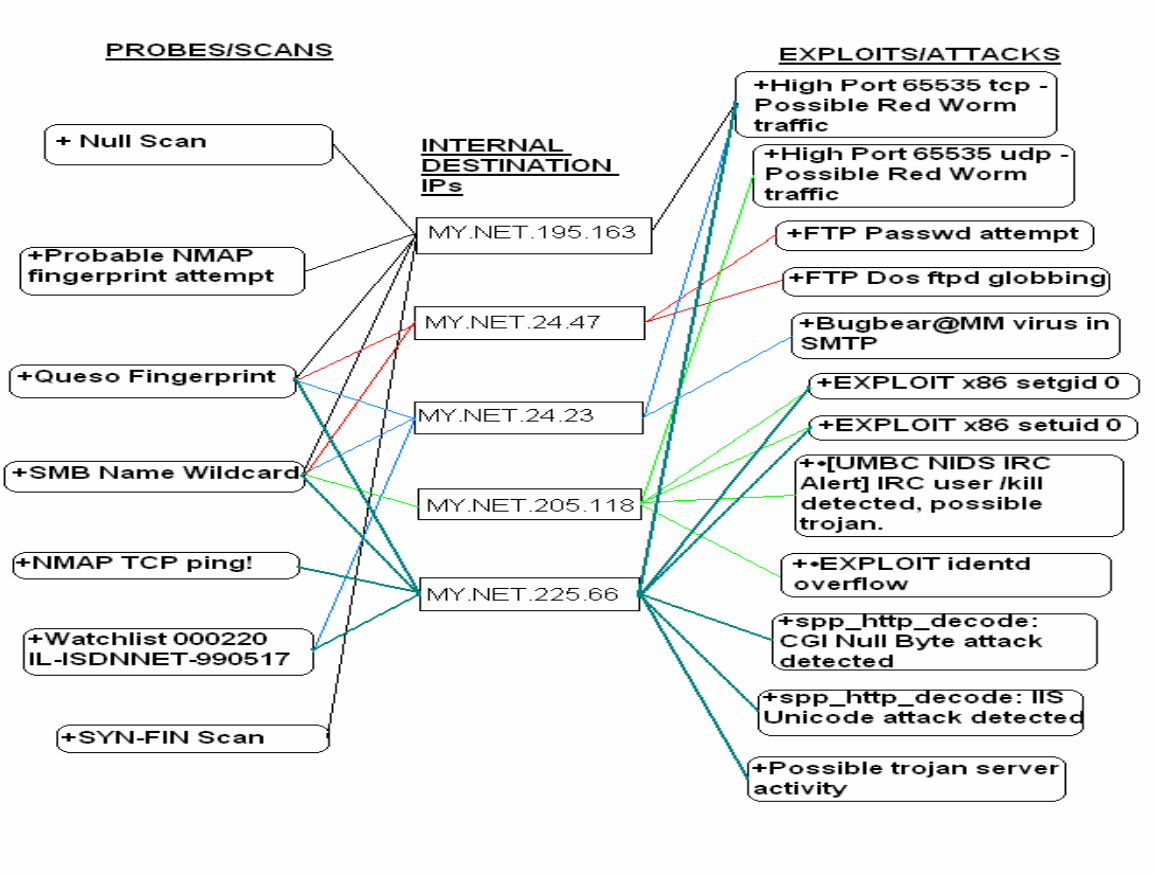
Assuming that both alerts are based on the same rule, the alerts would have to be classified as "Authentic". The reason for this is that the TCP flags that have to be set, to set off this rule are in an invalid combination, and would almost certainly have to be crafted. This alert is the basis for classifying 213.84.229.115 as having performed one of the most coordinated attacks, that and the other alerts it triggered also had their basis in TCP flag manipulation, especially those resulting in invalid TCP flag combinations.

131.118.254.130 generated four unique alerts which had very similar names, all beginning with the word "EXPLOIT x86". Searching the Snort alert documentation actually listed the alerts as "SHELLCODE x86". Reviewing these rules revealed that the traffic needed to set off these alerts would have to be specific for the x86 processor platform. This could mean that the attacker either had previous knowledge of the machine or is trying exploits meant for x86 blindly. We have to assume that this particular set of events is "Authentic". The knowledge required to perform these attacks would be significantly higher than that of the average user: shellcode, NOOP sleds, system calls. The "setgid 0" and "setuid 0" require knowledge of x86 assembly code to change the group and user id respectively, to those which have root privileges. The NOOP alerts also require knowledge x86 assembly code, specifically the NOOP code for x86, which when repeated the right amount of times and occupy the right amount of memory can constitute a buffer overflow attack. The variety yet homogeneity of these events suggest a coordinated and/or skilled attack.

Alert Link Graph

An alert link graph was created based on Internal Destination IPs so that the University's Systems Administrators and upper management could get an indication as to the complexity of attacker activity. The alerts were divided into two categories, Probes and Exploits. Probes being any activity meant to elicit a response from a server for information gathering purposes. Exploits being any activity that could potentially result in the compromise of the target system, command being executed on the target system, or the target system exhibiting symptoms of having software that is self-replicating or viral in nature. This link graph could be useful in ascertaining how their security devices should be configured and/or if any additional security measures need to be implemented.

Generally speaking, one can interpret the density of vertices converging on the Internal Destination IP nodes as correlating with complexity. The more dense a node is, the more complex attack is directed at it. From looking at the link graph, it is apparent that MY.NET.225.66 has been the target of a variety of probes AND exploits. Perhaps this is an indication that the subnet it is located on is not very well protected, but at least it seems the IDS is capable. MY.NET.24.47 was the target of two different FTP attacks, it should be looked into whether or not ftp is allowed for the department it belongs to and a temporary firewall rule be implemented to block ftp. If the link graph was generated for different subnets of MY.NET and it was noticed that some subnets contained much more "dense" nodes than others, this could be a sign of a weak link somewhere in their security structure. For example, from looking at the "PROBES/SCANS" nodes, "Queso Fingerprint" is one of the most popular type of probe/scans and has targeted various hosts on MY.NET. Queso scans are performed not surprisingly, with the Queso scanner, which tries to guess the OS of a target host by sending an assortment of valid and invalid tcp packets to first elicit a response from the target host and then compare those responses against a list of known responses for various operating systems. The Snort signature written to capture Queso traffic checks for the TCP flags ECN and CWR set, and for high TTL. Since the purpose of Queso scans is to discover remote host OS, one of the patterns that was looked for was OS specific attacks against the Queso scanned hosts. To be on the safe side, one must assume that there was a high probability that hosts that were scanned returned OS specific responses. Two of the hosts that were Queso scanned were hit with OS specific attacks. But they did not appear to be coordinated. Assuming that these hosts were hit with "Authentic" attacks, MY.NET.24.23 was hit with Red Worm and Bug Bear which are meant to attack Unix and Windows platforms respectively and MY.NET.225.66 was hit with attacks targeting IIS (which runs on Windows servers) and the EXPLOIT x86 attacks which target Linux systems. There are several possibilities to explain this: either the Queso scan returned an inconclusive response on OS detection, the attacker does not know their exploits, or the different attacks came from attackers which did not perform the Queso scan and were quite random.



Defensive Recommendations

Overall, there were several particular activities that warranted recommendations concerning security policy, security device configuration, and administration. One of these activities was the extensive and continuous activity of 213.84.229.115, which was able to scan 130.85.195.163 for thirteen hours straight and accounted for the top spot of 130.85.195.163 in the Top 10 Internal Destination IPs. A simple rule could have been placed in the router ACLs to stop this; for example:

```
access-list deny ip 213.84.229.115 0.0.0.0 any;
```

would be very effective. This also brings into question the human factor of security, if an attacker was able to scan for thirteen hours straight, what level of human interaction was present to defend the University's network, if any at all? It did not look like any action was taken on behalf of this activity. This could be due to several things. Perhaps there was no system of realtime alert, maybe logs were not reviewed on a regular basis. It doesn't take much to detect a scan of that scope and length, scripts can easily be written to generate some sort of real-time alert if a threshold is exceeded; there are also many free open-source tools and software available that can monitor as well. Because of this activity, it is recommended that the University implement as procedure the practice of reviewing system logs (firewall, syslog, etc) on a semi-hourly basis. If no one is able to

oversee the reviewing of the logs in off hours, than they should at least be reviewed on a daily basis.

Another activity that was of concern was that of two internal hosts, MY.NET.201.106 and MY.NET.251.70, which initiated connections to external hosts on a relatively unused and obscure port (TFTP, port 69). This activity brings into focus the egress security of the University (which could also easily be managed via ACLs and firewalls). This aspect of security is more difficult to implement than security that deals with ingress traffic, most people are concerned with attacks originating external to their home network and thus have their security software/devices set to detect for the most part incoming attacks. These hosts could have been participating in unauthorized transfer of sensitive information, but no logs were provided to confirm this. Another aspect of this activity that is worrisome is that the criticality of these two hosts was not known, since no configuration management documentation was provided to find out what their purposes were, one cannot say for sure if the data on those hosts needed to remain confidential (student records and personal information perhaps). Overall, to mitigate the risks associated with the above factors, the following defensive recommendations are proposed to the University:

- To address egress security - It is recommended that any suspicious egress activity be blocked egress at the router ACLs until the activity can be confirmed as legitimate or illegitimate (after regular log analysis has been established of course).
- To address unauthorized data transfer - Logs showing the amount of data transfer must be provided the next time analysis is performed to confirm illegitimate data transfer.
- To address host criticality issues - Configuration management documentation must be present and readily available to confirm whether questionable activity is legitimate for a particular host or set of hosts.

Regarding overall security of the University's network, two issues are addressed: reducing the volume of attacks and reducing the severity of attacks. Many of the actions recommended to reduce both go hand-in-hand. One of the activities that was noticed was that most of the internal ephemeral ports that were targeted were associated with well-documented applications including (gnutella, ms-sql server, http-proxy). If the University's perimeter security policy was of the type "Deny all not specifically allowed", the Internal Ephemeral ports list may have looked much different, consistent with that of established sessions. The ephemeral ports may have included more varied (and undocumented ports). Therefore it is recommended that the University implement a "Deny all not specifically allowed" perimeter policy regarding TCP and UDP ports; allowing ingress traffic destined for specifically unblocked ports. This can easily be done with a simple packet-filtering firewall. This action will help reduce the overall volume of scans and attacks and will also have an impact in reducing severity of attacks, especially against those ports which are blocked.

Regarding severity of attacks, many of the alerts triggered were for scans attempting to elicit a response from the target hosts, either to discover open ports or for remote OS detection (e.g., Queso fingerprint, SYN-FIN, Null scan) this is an important

part of an attacker's strategy, so that he/she may plan for the appropriate exploits. It is recommended that the University install on critical hosts, software that is discriminating in who and how it responds to hosts. This can include TCP wrappers which "wrap" around various services such as telnet or ssh, and introduce flexible access control. If the University's systems administrators are skilled enough, they can modify and/or tune the TCP/IP stacks on their critical hosts to limit how it responds to out-of-spec or otherwise invalid packets. This will make it harder to detect the OS. Programs such as ndd for Sun Solaris can be used to do this. For windows, certain TCP/IP parameters are accessible through the registry. It is recommended that the system administrator have expert knowledge of TCP/IP to do this.

In conclusion, all defensive recommendations are made in lieu of all available and provided data/logs. If all defensive recommendations are implemented, that University will have made significant steps in reducing the volume and severity of attacks.

Identification of Dangerous IPs

Several IPs were identified for further investigation. All registration information was obtained through www.arin.net and www.ripe.net where applicable. If the University decides to pursue further action, the following information is available (with explanation and registration information):

1. 63.231.14.237
   This address was deemed suspicious because an internal host had attempted to connect to it on TFTP (and subsequently alerted on "TFTP – Internal TCP connection to external tftp server"), a service not normally known for Internet-wide communications.

```
OrgName:    U S WEST Internet Services
OrgID:      USW
Address:    950 17th Street
Address:    Suite 1900
City:       Denver
StateProv:  CO
PostalCode: 80202
Country:    US

NetRange:   63.224.0.0 - 63.231.255.255
CIDR:       63.224.0.0/13
NetName:    USW-INTERACT99
NetHandle:  NET-63-224-0-0-1
Parent:     NET-63-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.USWEST.NET
NameServer: NS2.DNVR.USWEST.NET
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    1999-06-07
Updated:    2002-08-12

TechHandle: ZU24-ARIN
TechName:   U S WEST ISOps
TechPhone:  +1-612-664-4689
```

```
TechEmail:  abuse@uswest.net

OrgAbuseHandle: QIA2-ARIN
OrgAbuseName:   Qwest IP Abuse
OrgAbusePhone:  +1-703-363-3001
OrgAbuseEmail:  abuse@qwest.net

OrgNOCHandle: QIN-ARIN
OrgNOCName:   Qwest IP NOC
OrgNOCPhone:  +1-703-363-3001
OrgNOCEmail:  support@qwestip.net

OrgTechHandle: QIA-ARIN
OrgTechName:   Qwest IP Admin
OrgTechPhone:  +1-888-795-0420
OrgTechEmail:  ipadmin@qwest.com
```

2. 81.5.166.85

This address was also deemed suspicious because an internal host had also attempted to connect to it on TFTP (alerting as well on "TFTP – Internal TCP connection to external tftp server").

```
inetnum:      81.5.166.0 - 81.5.166.255
netname:      UK-ECLIPSE-ADSLSTATIC
descr:        ECLIPSE ADSL STATICIP
country:      GB
admin-c:      EIR3-RIPE
tech-c:       EIR3-RIPE
changed:      jim@eclipse.net.uk 20030320
notify:       as-guardian@eclipse.net.uk
mnt-by:       ECLINET-NMC
source:       RIPE
status:       ASSIGNED PA
remarks:      INFRA-AW
route:        81.5.128.0/18
descr:        Eclipse Networking Ltd.
origin:       AS12513
notify:       as-guardian@eclipse.net.uk
mnt-by:       ECLINET-NMC
changed:      jim@eclipse.net.uk 20020530
source:       RIPE
role:         Eclipse Internet - Ripe Admin
address:      Eclipse Internet,
address:      Portland House, Longbrook Street,
address:      Exeter, Devon EX4 6AB
address:      GB
phone:        +44 1392 333300
fax-no:       +44 1392 333310
e-mail:       support@eclipse.net.uk
trouble:      spam and abuse complaints =  mailto:abuse@eclipse.net.uk
admin-c:      JT5873-RIPE
tech-c:       JB15805-RIPE
tech-c:       GH9237-RIPE
tech-c:       JT5873-RIPE
nic-hdl:      EIR3-RIPE
```

```
remarks:      Eclipse Internet
notify:       jim@eclipse.net.uk
mnt-by:       ECLINET-NMC
changed:      jim@eclipse.net.uk 20020819
source:       RIPE
```

3. 146.164.34.42
   This external address alerted on scanning for a wide range of the University's Class B network on port 443.

```
OrgName:    Federal University of Rio de Janeiro
OrgID:      FURDJ
Address:    Nucleo de Computacao Eletronica
Address:     Caixa Postal 2324
Address:     CEP 20.001
Address:     Rio de Janeiro, RJ
City:
StateProv:
PostalCode:
Country:    BR

NetRange:   146.164.0.0 - 146.164.255.255
CIDR:       146.164.0.0/16
NetName:    REDE-UFRJ
NetHandle:  NET-146-164-0-0-1
Parent:     NET-146-0-0-0-0
NetType:    Direct Assignment
NameServer: ULTRIX1.NCE.UFRJ.BR
NameServer: CEOP1.REDERIO.BR
NameServer: NOC.CERF.NET
Comment:
RegDate:    1991-02-15
Updated:    1992-10-15

TechHandle: CM169-ARIN
TechName:   Mendes, Carlos
TechPhone:  +55 021 598-3118
TechEmail:  carlos@ceop1.rederio.br
```

4. 213.84.229.115
   This host warrants further investigation because it attempted what appeared to be a concerted effort to remotely identify the OS of MY.NET.195.163.

```
inetnum:      213.84.192.0 - 213.84.255.255
netname:      XS4ALL-ADSL
descr:        ADSL Static IP numbers
country:      NL
admin-c:      CB127
admin-c:      OD45
tech-c:       CB127
tech-c:       OD45
status:       assigned PA
remarks:      Please send email to "abuse@xs4all.nl" for complaints
remarks:      regarding portscans, DoS attacks and spam.
notify:       netmaster@xs4all.nl
```

```
mnt-by:       XS4ALL-MNT
changed:      oliver@xs4all.nl 20010711
changed:      oliver@xs4all.nl 20020710
source:       RIPE
route:        213.84.0.0/16
descr:        XS4ALL networking
origin:       AS3265
notify:       as-guardian@xs4all.nl
mnt-by:       XS4ALL-MNT
changed:      erik@xs4all.net 20000329
source:       RIPE
person:       Cor Bosman
address:      XS4ALL Internet BV
address:      Postbus 1848
address:      1000BV Amsterdam
address:      The Netherlands
phone:        +31 20 3987654
fax-no:       +31 20 3987601
e-mail:       cor@xs4all.net
nic-hdl:      CB127
mnt-by:       XS4ALL-MNT
changed:      cor@xs4all.nl 19980503
source:       RIPE
person:       Oliver Daudey
address:      XS4ALL Internet B.V.
address:      Eekholt 42
address:      1112 XH  Amsterdam
phone:        +31 20 3987654
fax-no:       +31 20 3987601
e-mail:       oliver@xs4all.nl
nic-hdl:      OD45
notify:       oliver@xs4all.nl
changed:      oliver@xs4all.nl 19980422
changed:      remcovz@xs4all.net 20010312
source:       RIPE
```

5. 131.118.254.130

This host warrants further investigation because it attempted what appeared to be a concerted effort to perform platform and OS specific attacks.

```
OrgName:    University of Maryland
OrgID:      UNIVER-270
Address:    System Administration
Address:    3300 Metzerott Road
City:       Adelphi
StateProv:  MD
PostalCode: 20783
Country:    US

NetRange:   131.118.0.0 - 131.118.255.255
CIDR:       131.118.0.0/16
NetName:    MINCNET
NetHandle:  NET-131-118-0-0-1
Parent:     NET-131-0-0-0-0
NetType:    Direct Assignment
NameServer: NS.USMD.EDU
```

```
NameServer: UMCPNOC.UMS.EDU
NameServer: NOC.USMD.EDU
NameServer: TRANTOR.UMD.EDU
Comment:
RegDate:    1988-11-15
Updated:    1998-11-24
TechHandle: NM162-ARIN
TechName:   Malmberg, Norwin
TechPhone:  +1-301-445-2758
TechEmail:  malmberg@usmh.usmd.edu
```

Overall Recommendations

This analysis would have been much more complete if the University had provided full packet captures, network topology, and configuration documentation. Analysis was performed to the best of the analyst's ability given the limited data set. It is strongly recommended that the University review their security policy and procedures as well as their perimeter security.Hopefully, next time an audit is performed all needed data will be provided.

Java Programs

The following is a Java program developed to parse through the alert files and calculate various statistics. The programs written to parse through the scans and oos files are very similar and thus were not included due to length.

```
import java.io.*;
import java.util.*;

public class Alert
{


public static void main(String[] args)
{
String  timestamp, misc1, alertName, src, dst, srcIP, dstIP, temp;
String line, service, src_port, length, rule,tempKey,tempKeyValue;

int count=0, alertCount=0;
Integer event_Count =  new Integer(0), tempInt = new Integer(0);

ScansObject[] alertArray;
Vector theVector;

Hashtable alertHash = new Hashtable();
Hashtable intSrcAlertIPHash = new Hashtable();
Hashtable extSrcAlertIPHash = new Hashtable();
Hashtable intDstAlertIPHash = new Hashtable();

Collection values;
Iterator keyIter, valueIter;
Set keys;
```

```
try
{
      FileReader fr = new FileReader(args[0]);
      BufferedReader br = new BufferedReader(fr);
      line = br.readLine();


while(line!=null)
{

      StringTokenizer str = new StringTokenizer(line);

      if(str.countTokens() < 4)
      {
            line = br.readLine();
            continue;
      }

timestamp = str.nextToken();
misc1 = str.nextToken();
alertName = getAlertName(str);

      if( !(alertName.equalsIgnoreCase("portscan")) )
      {
            if(str.hasMoreTokens())
            {
                  src = str.nextToken();
                  StringTokenizer ipaddress = new StringTokenizer(src,
":");
                  srcIP = ipaddress.nextToken();


                    if(srcIP.startsWith("MY.NET"))
                  {
                        if(!intSrcAlertIPHash.containsKey(srcIP))
                        {
                              theVector = new Vector(20);
                              theVector.addElement((String)alertName);
                              intSrcAlertIPHash.put(srcIP, theVector);
                        }
                        else if(intSrcAlertIPHash.containsKey(srcIP) )
                        {
                                    theVector = (Vector)
intSrcAlertIPHash.get(srcIP);

                              if(!theVector.contains( (String)
alertName) )
                              {
                                    theVector.addElement( (String)
alertName);
                                    intSrcAlertIPHash.put(srcIP,
theVector);
                              }
                        }
                  }
                  else
                  {
```

```
                        if(!extSrcAlertIPHash.containsKey(srcIP))
                        {
                              theVector = new Vector(20);
                              theVector.addElement((String)alertName);
                              extSrcAlertIPHash.put(srcIP,theVector);
                        }
                        else if(extSrcAlertIPHash.containsKey(srcIP) )
                        {
                              theVector = (Vector)
extSrcAlertIPHash.get(srcIP);

                              if(!theVector.contains( (String)
alertName) )
                              {
                                    theVector.addElement( (String)
alertName);
                                    extSrcAlertIPHash.put(srcIP,
theVector);
                              }
                        }
                  }

            }

            if(str.hasMoreTokens())
            {
                  str.nextToken();
            }

            if(str.hasMoreTokens())
            {
                  dst = str.nextToken();
                  StringTokenizer ipaddress = new StringTokenizer(dst,
":");
                  dstIP = ipaddress.nextToken();

                        if(dstIP.startsWith("MY.NET"))
                        {

if(!intDstAlertIPHash.containsKey(dstIP))
                                    {
                                          theVector = new Vector(20);

theVector.addElement((String)alertName);
                                          intDstAlertIPHash.put(dstIP,
theVector);
                                    }
                                    else
if(intDstAlertIPHash.containsKey(dstIP) )
                                    {
                                          theVector = (Vector)
intDstAlertIPHash.get(dstIP);

                                          if(!theVector.contains(
(String) alertName) )
                                          {
```

```
                                                    theVector.addElement(
(String) alertName);

intDstAlertIPHash.put(dstIP, theVector);
                                                }
                                    }
                          }


               }

         }


         if(!alertHash.containsKey(alertName))
           {
                alertCount++;
                event_Count = new Integer(1);
                alertHash.put(alertName, event_Count);
                }
          else if(alertHash.containsKey(alertName))
          {
                count = 0;
                count = ((Integer)alertHash.get(alertName)).intValue();
                count+=1;
                event_Count = new Integer(count);
                alertHash.put(alertName,event_Count);
          }


line = br.readLine();

}//end while
}
catch(FileNotFoundException e)
{
 System.out.println("Snort alert file unreadable!");
}
catch(IOException e)
{
 System.out.println(e);
}

/********
debug
*******/
System.out.println("Number of Unique Alerts: " + alertCount);


keys = alertHash.keySet();
keyIter = keys.iterator();
alertArray = new ScansObject[alertCount];

for(int i = 0; (i < alertCount) && keyIter.hasNext(); i++ )
{
      tempKey = (String)keyIter.next();
      tempInt = (Integer) alertHash.get(tempKey);
```

```
      count = tempInt.intValue();
      alertArray[i] = new ScansObject(tempKey, count);
}

/**********************************
*Sort the top 10
**********************************/
int k;

for(int p=0; p < alertArray.length; p++)
{
      ScansObject tmp = alertArray[p];

      for(k = p; k > 0 && tmp.compareTo(alertArray[k-1]) < 0; k--)
            alertArray[k] = alertArray[k-1];

      alertArray[k] = tmp;
}

for(int i = (alertArray.length - 1); i > (alertArray.length - 12); i--)
      System.out.println(alertArray[i].theObject + " " +
alertArray[i].count + "\n");


/***************
Internal Source IP Alerts
****************/
System.out.println("****************************\n*******INTERNAL
SOURCE IP ALERTS*******");
keys = intSrcAlertIPHash.keySet();
keyIter = keys.iterator();

while(keyIter.hasNext())
{
        tempKey = (String)keyIter.next();
        theVector = (Vector)intSrcAlertIPHash.get(tempKey);

        if(theVector.size() >= 2)
        {
                System.out.println("\nAlerts for INTERNAL SOURCE IP
address " + tempKey);
                Iterator theIterator = theVector.iterator();

                while(theIterator.hasNext() )
                {
                        System.out.println("    " + ((String)
theIterator.next()) );
                }
        }

}

/*****************
External Source IP Alerts
******************/
System.out.println("****************************\n*******EXTERNAL
SOURCE IP ALERTS*******");
```

```
keys = extSrcAlertIPHash.keySet();
keyIter = keys.iterator();

while(keyIter.hasNext() )
{
        tempKey = (String)keyIter.next();
        theVector = (Vector)extSrcAlertIPHash.get(tempKey);

      if(theVector.size() >= 2)
      {
              System.out.println("\nAlerts for EXTERNAL SOURCE IP
address " + tempKey);
              Iterator theIterator = theVector.iterator();

            while(theIterator.hasNext() )
            {
                  System.out.println("    " + ((String)
theIterator.next()) );
            }
      }
}


/*****************
Internal Destination IP Alerts
******************/
System.out.println("****************************\n*******INTERNAL
DESTINATION IP ALERTS*******");
keys = intDstAlertIPHash.keySet();
keyIter = keys.iterator();

while(keyIter.hasNext() )
{
        tempKey = (String)keyIter.next();
        theVector = (Vector)intDstAlertIPHash.get(tempKey);

        if(theVector.size() >= 2)
        {
                System.out.println("\nAlerts for INTERNAL DESTINATION
IP address " +
tempKey);
                Iterator theIterator = theVector.iterator();

                while(theIterator.hasNext() )
                {
                        System.out.println("    " + ((String)
theIterator.next()) );
                }
        }
}


}//end main



public static String getAlertName(StringTokenizer str)
```

```
{

String temp="", alertName="";

temp = str.nextToken();

      if(temp.startsWith("spp_portscan"))
      {
            return "portscan";
      }
      else
      {
            while(str.hasMoreTokens() )
            {
                  alertName = alertName.concat(temp + " ");
                  temp = str.nextToken();
                  if(temp.startsWith("[**]"))
                  {
                        return alertName;
                  }

            }
      }
return alertName;
}


}//end class
```