Global Information Assurance Certification Paper

GIAC GCIA Practical version 3.3

The PacketShaper's® Impact on Intrusion Detection Practice

Carl Gibbons

4 February 2003

Part 1

The PacketShaper's® Impact on Intrusion Detection Practice

Contents:

Introduction

Packeteer®, Inc. (Packeteer[1]), a seven-year-old company headquartered in Cupertino, California, markets a *quality of service (QoS)* network device called *PacketShaper*. It is a bandwidth management tool which has become very popular and widely deployed. The device is commonly placed at a network's perimeter, to manage Internet traffic. Its operating system software, called *PacketWise*™, detects TCP and other protocols' flows, tries to identify the service represented by each data flow, and allocates bandwidth among the flows, based on policies configured by an administrator. To accomplish its task, it actually alters certain header fields of some of the packets it manages.

This paper addresses some of the ways this device affects a security professional's work. I hope to:

1. explain PacketShaper workings to security professionals, and packet analysts in particular,
2. assist intrusion analysts who, like me, not only use a NIDS (network intrusion detection system) but also use their "packet jockey" skills to care for a PacketShaper,
3. show how the PacketShaper complements a NIDS.

Many of the same traffic identification techniques used by intrusion analysts are used internally by the PacketShaper, other Packeteer products, and other vendors' QoS products. Intrusion detection efforts may be enhanced when these tools augment analysts arsenals. Along with firewalls and routers, they are another element of an organization's network countermeasures against undesirable traffic. Also, when analyzing network traces, an analyst should have a correct understanding of the PacketShaper's mechanisms, and how it perturbs traffic.

Other traffic management products on the market include Sitara Networks® *QoSWorks* appliance (Sitara Networks), Lightspeed Systems *Total Traffic Control* software (Lightspeed Systems), and Allot Communications *NetEnforcer*® devce (Allot Communications). There are also efforts to provide open-source QoS technologies in the Linux kernel. (Dawson) I have never used any of these, so this paper centers on the product familiar to me, the PacketShaper. Nevertheless, some of the intrusion detection issues examined here may be relevant to environments where any of these products are deployed.

Before examining the impact of this device on intrusion detection, let's consider where devices like the PacketShaper fit into the bigger picture.

Background Material

Definitions of Quality of Service

The acronym *QoS* refers generally to data delivery protocols or technologies with the following properties:

1. they identify and prioritize different kinds of transmitted data by some kind of distinguishing characteristic,
2. they use these priorities to decide how to deliver different kinds of traffic, and
3. they are judged by how well they ensure proper delivery of higher priority data.

In real world networks, concrete goals must stand in the place of these abstract properties. Such goals answer corresponding questions:

1. What applications have higher priority network traffic? What characteristics distinguish these applications' data?
2. How should the high priority data be delivered? (In rapid bursts? In steady, more predictable flows?)
3. What constitutes acceptable results? (Fastest possible delivery? No lost or retransmitted packets?)

Eric D. Siegel defines QoS as "a somewhat vague term referring to the technologies that classify network traffic and then ensure that some of that traffic receives special handling." (Siegel, p 4)

The definition of QoS is also subject to perspective. William C. Hardy defines both *intrinsic* and *perceived* QoS, where the former represents the point of view of the designer or operator of a network, and the latter the point of view of the network's users and customers. He cites as an example his work with a certain voice network. It had great *intrinsic QoS*, with fidelity so high that background hiss was virtually eliminated. But it had poor *perceived QoS*, because without some background hiss the users often mistook silence as a sign of disconnected calls. (Hardy, p 6) This is a useful distinction to consider in data networks as well. Different users have different opinions about what kind of traffic is important. Quality of service may become a zero-sum game, in which one application improves only at the expense of another application. Peer-to-peer file traders and online computer game players usually find their quality of network service diminish after network administrators deploy a PacketShaper and configure it to throttle or deny their bandwidth-hungry applications.

QoS also refers to academic studies surrounding these principles and technologies. For example, traffic marking, queuing, availability measurements, and service level guarantees are all studied in QoS circles. It's a very broad field. With so many approaches and facets that may affect intrusion detection work, let's narrow our focus, beginning with an oversimplified characterization of the problem that a PacketShaper solves.

Yet Another ``Tragedy of the Commons''

By default, flows over a TCP/IP network are treated equally; that is, routers, switches, and other network devices do their best to deliver all pieces of data to their intended destinations, without discrimination. The literature calls this "best effort" service. (Kilkki, pp 44-53; Siegel, p 3)

Example: best effort gives leisure web surfing the same priority as revenue-generating secure transactions. What happens when there aren't enough bits-per-second for both bandwidth-hungry recreational applications and mission critical traffic?

The "Tragedy of the Commons" (Disenchanted) is an economic parable about a green field where everybody may graze their livestock toll-free, so everybody does, and overgrazing spoils the common field for everybody. When competing applications exhaust their limited shared bandwidth, we call it network congestion. It is the tragedy of the commons played on an electronic stage: "bandwidth overgrazed." Demand exceeds supply. Packets get dropped. Packets need to be retransmitted. Effective throughput of the congested network decreases. With all traffic getting best-effort service, the network pipe has been ruined for everybody.

The ``Commons'' Under New Management

One escape from the tragedy is overprovisioning (Siegel, 193), which means, purchase more bandwidth. This is the least complicated solution, but usually very expensive.

To continue our rudimentary point of view: instead of treating all packets as equals, find a way to determine which packets are mission critical communications and which are optional luxuries, and when facing competitive congestion, deliver the important packets and delay the unimportant ones. This is what the PacketShaper's traffic shaping technology accomplishes. Bandwidth investments, under this approach, grow proportionally to the needs of mission critical applications, instead of in proportion to the burgeoning appetites of all applications combined. Therefore, budget forecasters tend to find this approach more cost effective than overprovisioning. (This may explain Packeteer's uncharacteristic corporate success compared to other Internet technology companies' struggles during the recent market downturn.)

Real-world networks have much more complicated situations than this oversimplified good-traffic versus bad-traffic example. Nevertheless, the principles illustrated in the example are sound.

Getting Nitty-Gritty With the PacketShaper

Where the Wild Things Are

A PacketShaper is inserted in the path of the network traffic to be managed. One of its interfaces is oriented to the "outside" network, the other "inside." Michael Nancarrow expressed a fear shared by many that the PacketShaper potentially introduces "another point of failure" in the network stream. (Nancarrow) This fear of a layer-one security vulnerability is utterly unfounded. Unlike a firewall, which "fails closed," a PacketShaper "fails open." When shut down or powered off, the PacketShaper's interfaces close their circuits into a "hardware pass-through" mode. The machine can actually be unplugged without disrupting connectivity.

The first property of QoS, identification and classification, is handled automatically by the PacketShaper's *Traffic Discovery* feature. Traffic Discovery mode can be toggled on or off. (Packeteer also sells a similar product called *PacketSeeker* that also discovers and classifies traffic, but does not control flows.)

Internally, a PacketShaper keeps a data structure of discovered and configured traffic classes, called a *traffic tree*. Here's how this data structure is represented visually in the device's web user interface:

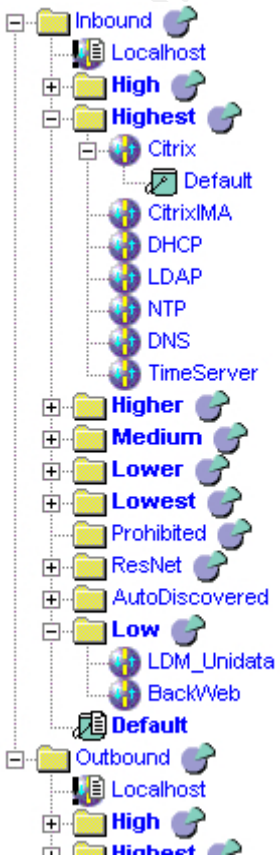


Figure: PacketShaper traffic tree

It is actually two trees; one root class is called `/Inbound` and the other `/Outbound`. Each flow falls into exactly one of these two categories, depending on whether an outside or inside host initiated the flow. Under these two base traffic classes fall other subclasses, or hierarchies of classes. Also, there is an "Enable Discovery" attribute that can be cleared or set on a class to make that class *discovery enabled*. It is set by default for the two root classes.

When Traffic Discovery mode is turned on, new subclasses automatically appear under discovery enabled classes, as traffic is inspected and identified. For example, after it sees an inside host exchange a few NTP datagrams with an outside time server, a new subclass `/Outbound/NTP` will be created under a discovery enabled `/Outbound` base class. Traffic that the PacketShaper does not know how to classify goes into a subclass named `Default`.

I have found it useful to turn off the Enable Discovery toggle in the root classes, and instead create discovery enabled `/Inbound/AutoDiscovered` and `/Outbound/AutoDiscovered` subclasses with a rule to match all IP traffic. Then when I see a class appear that I want to manage, such as `/Outbound/AutoDiscovered/NTP`, I use the PacketShaper's web user interface to move the subclass to another part of the `/Outbound` class tree. Also, with this configuration the PacketShaper never tries to classify non-IP traffic, but instead just relegates it to the `/Inbound/Default` and `/Outbound/Default`.

The Traffic Discovery feature provides empirical evidence of the types of traffic that actually flow over a network. This is handy data for configuring or tuning the rule-sets/signatures of an intrusion detection system. For example, an intrusion analyst may have disabled a rule that generates alerts for a traffic type deemed not applicable to the protected network. If the PacketShaper discovers that traffic type, it may induce the analyst to re-enable that rule.

The white paper *TCP Rate Control and Alternatives* lists several criteria that a PacketShaper (or PacketSeeker) can use to identify and classify traffic. (Packeteer[2], p 7-8) Some of these criteria have counterparts in tcpdump filter expressions (also known as BPF filters) used by packet analysts to identify traffic. The following table shows some examples:

Packet property:	**Possible corresponding tcpdump filter primitives:**
Protocol	`ip`, `arp`, `icmp`, `tcp`, `udp`, `ip proto`

IP precedence bits	`ip[1]`
Port number	`port`
IP address	`host`
MAC address	`ether host`
Source/destination	`src, dst`
Subnet	`net`
Travel direction (inbound/outbound)	`ether dst, ether src`

However, a PacketShaper is also able to delve into packet's payloads and decode many application layer protocols. This is sometimes called "layer seven" traffic inspection, referring to all possible layers in the OSI model. (Jupitermedia) Some of these traffic properties include URLs, mime types, Oracle databases, and Citrix published applications. These are difficult or impossible to identify using tcpdump filter expressions, but there are other packet analysis tools such as Ethereal (Ethereal) that employ application layer decoding technology.

John Cupps noted the advantage of application layer identification in managing bandwidth-hungry peer-to-peer music sharing traffic: "...through the use of seven-layer classifications, Packetshaper limits incoming and outgoing traffic traveling through dynamically assigned ports. MP3s, the media file type used with Napster, requires this seven-layer classification..." (Cupps)

Furthermore, the PacketShaper notices when uncharacteristic traffic travels over a well known port. For example, it only classifies a TCP flow over port 80 as part of the `HTTP` subclass if it notices HTTP protocol in the application layer. If it finds something other than HTTP flowing over port 80, it classifies it differently. For example, I have seen Traffic Discovery create a subclass `Discovered_Ports/TCP_Port_80` for port 80 traffic that it did not know how to classify. Also, it can detect some applications that encapsulate or tunnel their data within HTTP, and classify that traffic by the application that produced it instead of lumping it in with the rest of the `HTTP` class. An example of this is traffic produced by recent versions of Kazaa. (Sharman Networks)

Also, it follows port hopping applications. These start communicating over some port, but during a conversation make new connections over other ports or protocols to continue their transactions. Examples include FTP (Bell) and RealNetworks (RealNetworks) products. It can even follow transactions that try and hide by hopping to unpredictable ports (but it does not always succeed in following some port hopping applications).

A PacketShaper can also differentiate traffic from hosts in different speed ranges; for example, it can be configured to detect dial-up analog modem users and

classify their traffic separately from high-speed hosts.

Finally, other QoS technologies such as Differentiated Services (IETF[1]; see also Kilkki) and MPLS (IETF[2]) may be used to mark packets for special handling by switches and routers. A PacketShaper can identify and classify traffic by these marks as well, and can also participate on a class-by-class basis by setting diffserv code values in IP headers' TOS bytes.

It's appropriate to declare at this point that the PacketShaper is not an intrusion detection system, even though it sports this wide array of traffic recognition abilities that are also attractive in an IDS. Nevertheless, these abilities can complement an IDS, by filtering or focusing the traffic scenarios an intrusion analyst might want to investigate.

The PacketShaper as a ``Message Massager''

Traffic management corresponds to the second property of QoS, prioritized data delivery. The primary method used by the PacketShaper to control identified and classified flow is *traffic shaping*, or *rate control* of TCP connections. The technique only applies to TCP traffic, but since the vast majority of Internet traffic uses TCP anyway, it is very effective.

To understand this technique, let's review the "best effort" flow control mechanisms of TCP, *sliding windows* and *slow start*.

Two bytes at offset 14 of every TCP segment header advertize a dynamic window size, measured in bytes. This is how the receiver tells the sender how much buffer space it has available. As the buffer fills, the window size narrows. If the buffer is exhausted, the window size is zero, and the sender must stop transmitting. As the receiver processes the data in the buffer, buffer space is freed, the window size widens, and the sender may transmit more data.

Slow start refers to the rate at which the receiver acknowledges transmitted TCP segments. After the three-way handshake that starts a connection, the sender transmits only one segment and waits for an acknowledgement. After that, the sender may trasmit two segments, and wait for another acknowledgement. If all went well, the sender may transmit three or perhaps four segments before waiting for the acknowledgement. The sender keeps increasing and measuring the number of segments it transmits between acknowledgements, until the receiver acknowledges none or only some of a transmission; then the sender reduces the amount of data it sends between acknowledgements, to avoid causing network congestion. For a thorough discussion of sliding windows and slow start, see Stevens' chapter on *TCP Bulk Data Flow*. (Stevens, pp 275-286)

Despite the adjective in the term "slow start," best-effort delivery of TCP segments tends to ramp up bandwidth usage very quickly; after all, the

algorithms work at high CPU and network speeds.

To regulate this phenomenon, the PacketShaper alters the sliding windows in TCP flows. It intercepts every data receiver's acknowledgement packet, and if it determines that the window size in the packet would cause a bandwidth hogging burst from the sender, or in other words, the window size is too large for a properly paced transmission, it crafts a new acknowledgement packet with a fraction of the original's window size. It passes that crafted acknowledgement along to the sender instead. The sender transmits enough segments to fill the reduced advertized window, and the PacketShaper detects this flow. Then after a measured delay determined by a predictive scheduling algorithm, it crafts another acknowledgement packet for the sender. The sender transmits more data, and the PacketShaper measures another delay, and the process continues. The *TCP Rate Control and Alternatives* white paper discusses this algorithm in more detail. (Packeteer[2], pp 8-10) Packeteer holds a patent on the mechanism.

Siegel provides the following illustration. It depicts a visual example of four acknowledgements with 2000-byte window sizes, crafted by a PacketShaper to replace and pace an 8000-byte advertized window size. He also notes that "rate control is considered by some to meddle with the operation of TCP in a way that violates the spirit of the TCP specification. Nevertheless, it is in widespread and successful use." (Siegel, p 187)

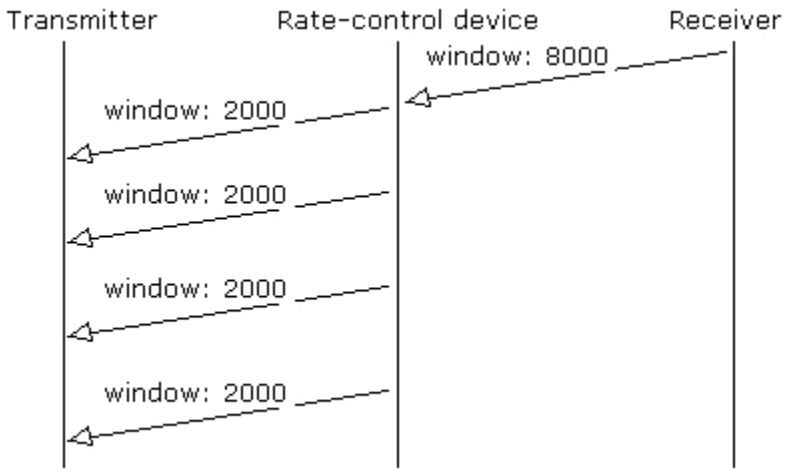


Figure: TCP rate control illustration (Siegel, p187)

The PacketShaper uses other methods to regulate non-TCP protocols. Another Packeteer white paper, *UDP Traffic Management* (Packeteer[3]), has some discussion on this topic.

Siegel described *bucket techniques* for queuing and regulating traffic. One of these is *"leaky bucket:"*

> "The leaky bucket can be used to create a constant-rate traffic flow from a varying rate flow, as the input rate to the leaky bucket can vary, but the output rate of the leaky bucket is a constant. The capacity of the bucket determines how much excess data can be buffered before data packets must be discarded." (Siegel, p 184)

He also explains about a variation called *token bucket*, (Siegel, pp 184-185), and reports that "stand-alone boxes, such as those from ... Packeteer, implement traffic shaping using the bucket technologies as well as by using class-based shaping or rate control." (Siegel, p 252)

Bandwidth Allocation

Just as the PacketShaper implements the first principle of QoS with its Traffic Discovery mode, it implements the second principle with its Traffic Shaping mode, which may also be toggled on or off.

For each class in the traffic tree, two mechanisms are available to control traffic matching that class and limit its bandwidth. When Shaping mode is turned on, the PacketShaper enforces these mechanisms' controls on each flow.

The first is called a *policy*. If a policy is set for a class, that policy applies to every traffic flow belonging to that class. There are five policies to choose:

- **Discard:** disallow that traffic by throwing away all packets belonging to that class.
- **Never-Admit:** disallow that traffic but return an error code, or in the case of web traffic, redirect it.
- **Ignore:** allow that traffic and don't meddle with it.
- **Priority:** allow that traffic and rank its importance. Priorities range between 0 (lowest) and 7 (highest). If a policy is not set for a class, it inherits a Priority 3 policy by default.
- **Rate:** allow that traffic and control it so that bursty traffic is smoothed. Rate policy settings may include a guaranteed rate (minimum bandwidth), a priority (0-7) at which flows' bursts may obtain more bandwidth, and a burst limit (maximum bandwidth).

The other mechanism is called a *partition*. These do not affect flows individually, but instead guarantee and/or limit the bandwidth of all flows in a class (or of all flows in a subtree of classes). Because partitions are specified using two rates, a guarantee (minimum) and a limit (maximum), they are easily confused with rate policies. Remember that policies apply to individual flows, but partitions control the aggregate of all flows in a class. The base classes `/Inbound` and `/Outbound` have default partitions with interesting values. The default guarantee rate is "uncommitted," meaning that traffic in the class is not guaranteed any bandwidth, but may receive whatever portion isn't already committed to other

traffic classes. The default limit rate is "none," meaning that aggregate traffic speed is not limited, except by the maximum speed allowed by the network or medium.

Experiments with tcpdump Reveal the Effects of Rate Policies

I set up an experiment to see the effects of traffic shaping at the packet level, and noticed some patterns that analysts may use to identify PacketShaper adjusted traffic. For a baseline trace of default "best-effort" traffic behavior, I set up two machines on an "Inside" LAN. One computer was a Unix FTP server, and the other was a laptop running the Windows 98 command line FTP client. I transferred a file from and to (using ftp commands `GET` and `PUT`) the ftp server, and recorded the transfers at both ends, using tcpdump version 3.7.1 on the server and WinDUMP version 3.6.2 on the laptop.

Note: in the WinDUMP traces that follow, the timestamps are occasionally skewed by slightly more than nine hours. I cannot explain these random hiccups in timestamps, but others have noted irregularities in WinDUMP timestamps as well. For example, see Matt Scarborough's response to Robin Stubbs' November 2001 "windump timezone gotcha" thread on the intrusions@incidents.org mailing list. (Scarborough; Stubbs)

Here is what the GET transfer looked like to the laptop's WinDUMP:

```
01:44:18.570220 ftpserver.20 > laptop.1044: S 362227150:362227150(0)
win 24820 <mss 1460> (DF)
10:46:39.591066 laptop.1044 > ftpserver.20: S 976616:976616(0) ack
362227151 win 9324 <mss 1332> (DF)
10:46:39.592203 ftpserver.20 > laptop.1044: . ack 1 win 25308 (DF)
10:46:39.599155 ftpserver.20 > laptop.1044: P 1:1333(1332) ack 1 win
25308 (DF)
01:44:18.711256 laptop.1044 > ftpserver.20: . ack 1333 win 9324 (DF)
10:46:39.736799 ftpserver.20 > laptop.1044: . 1333:2665(1332) ack 1 win
25308 (DF)
10:46:39.740415 ftpserver.20 > laptop.1044: P 2665:3997(1332) ack 1 win
25308 (DF)
10:46:39.742175 laptop.1044 > ftpserver.20: . ack 3997 win 9324 (DF)
10:46:39.747158 ftpserver.20 > laptop.1044: . 3997:5329(1332) ack 1 win
25308 (DF)
10:46:39.750754 ftpserver.20 > laptop.1044: . 5329:6661(1332) ack 1 win
25308 (DF)
10:46:39.754141 ftpserver.20 > laptop.1044: P 6661:7993(1332) ack 1 win
25308 (DF)
10:46:39.755884 laptop.1044 > ftpserver.20: . ack 7993 win 9324 (DF)
10:46:39.820269 ftpserver.20 > laptop.1044: . 7993:8193(200) ack 1 win
25308 (DF)
10:46:39.822438 ftpserver.20 > laptop.1044: . 8193:9525(1332) ack 1 win
25308 (DF)
10:46:39.825813 ftpserver.20 > laptop.1044: . 9525:10857(1332) ack 1
win 25308 (DF)
10:46:39.829180 ftpserver.20 > laptop.1044: . 10857:12189(1332) ack 1
```

```
win 25308 (DF)
10:46:39.830926 laptop.1044 > ftpserver.20: . ack 12189 win 9324 (DF)
...
10:46:39.954086 laptop.1044 > ftpserver.20: . ack 34101 win 1132 (DF)
01:44:18.936347 laptop.1044 > ftpserver.20: . ack 34101 win 5512 (DF)
01:44:18.941262 ftpserver.20 > laptop.1044: . 34101:35433(1332) ack 1
win 25308 (DF)
10:46:39.965262 ftpserver.20 > laptop.1044: . 35433:36765(1332) ack 1
win 25308 (DF)
10:46:39.968634 ftpserver.20 > laptop.1044: . 36765:38097(1332) ack 1
win 25308 (DF)
10:46:39.971301 ftpserver.20 > laptop.1044: F 38097:38797(700) ack 1
win 25308 (DF)
10:46:39.972403 laptop.1044 > ftpserver.20: . ack 38798 win 816 (DF)
01:44:18.954401 laptop.1044 > ftpserver.20: . ack 38798 win 5196 (DF)
01:44:18.993511 laptop.1044 > ftpserver.20: . ack 38798 win 9324 (DF)
01:44:19.016312 laptop.1044 > ftpserver.20: F 1:1(0) ack 38798 win 9324
(DF)
01:44:19.017326 ftpserver.20 > laptop.1044: . ack 2 win 25308 (DF)
```

and to the server's tcpdump:

```
10:46:51.701738 ftpserver.20 > laptop.1044: S 362227150:362227150(0)
win 24820 <mss 1460> (DF)
10:46:51.702991 laptop.1044 > ftpserver.20: S 976616:976616(0) ack
362227151 win 9324 <mss 1332> (DF)
10:46:51.703211 ftpserver.20 > laptop.1044: . ack 1 win 25308 (DF)
10:46:51.706364 ftpserver.20 > laptop.1044: P 1:1333(1332) ack 1 win
25308 (DF)
10:46:51.843493 laptop.1044 > ftpserver.20: . ack 1333 win 9324 (DF)
10:46:51.843991 ftpserver.20 > laptop.1044: . 1333:2665(1332) ack 1 win
25308 (DF)
10:46:51.844189 ftpserver.20 > laptop.1044: P 2665:3997(1332) ack 1 win
25308 (DF)
10:46:51.854067 laptop.1044 > ftpserver.20: . ack 3997 win 9324 (DF)
10:46:51.854360 ftpserver.20 > laptop.1044: . 3997:5329(1332) ack 1 win
25308 (DF)
10:46:51.854544 ftpserver.20 > laptop.1044: . 5329:6661(1332) ack 1 win
25308 (DF)
10:46:51.854707 ftpserver.20 > laptop.1044: P 6661:7993(1332) ack 1 win
25308 (DF)
10:46:51.867863 laptop.1044 > ftpserver.20: . ack 7993 win 9324 (DF)
10:46:51.868091 ftpserver.20 > laptop.1044: . 7993:8193(200) ack 1 win
25308 (DF)
10:46:51.868244 ftpserver.20 > laptop.1044: . 8193:9525(1332) ack 1 win
25308 (DF)
10:46:51.868408 ftpserver.20 > laptop.1044: . 9525:10857(1332) ack 1
win 25308 (DF)
10:46:51.868585 ftpserver.20 > laptop.1044: . 10857:12189(1332) ack 1
win 25308 (DF)
10:46:51.942897 laptop.1044 > ftpserver.20: . ack 12189 win 9324 (DF)
...
10:46:52.065871 laptop.1044 > ftpserver.20: . ack 34101 win 1132 (DF)
10:46:52.068664 laptop.1044 > ftpserver.20: . ack 34101 win 5512 (DF)
10:46:52.069006 ftpserver.20 > laptop.1044: . 34101:35433(1332) ack 1
win 25308 (DF)
```

```
10:46:52.069361 ftpserver.20 > laptop.1044: . 35433:36765(1332) ack 1
win 25308 (DF)
10:46:52.069624 ftpserver.20 > laptop.1044: . 36765:38097(1332) ack 1
win 25308 (DF)
10:46:52.069835 ftpserver.20 > laptop.1044: F 38097:38797(700) ack 1
win 25308 (DF)
10:46:52.084172 laptop.1044 > ftpserver.20: . ack 38798 win 816 (DF)
10:46:52.086411 laptop.1044 > ftpserver.20: . ack 38798 win 5196 (DF)
10:46:52.125646 laptop.1044 > ftpserver.20: . ack 38798 win 9324 (DF)
10:46:52.148598 laptop.1044 > ftpserver.20: F 1:1(0) ack 38798 win 9324
(DF)
10:46:52.148898 ftpserver.20 > laptop.1044: . ack 2 win 25308 (DF)
```

Except for the befuddled WinDUMP timestamps, the traces match up perfectly, and demonstrate TCP's "slow start" algorithm: one segment and an acknowledgement, two more segments and an ack, three and an ack, etc. The transactions are rapid, too: less than 80 milliseconds between successive laptop's acknowledgements. By the time we reach the end of the transfer, we see the sliding window algorithm demonstrated: the laptop acknowledges progressively larger windows as it processes the data it has received. The FTP `PUT` behaved predictably similarly.

Next, I created classes `/Inbound/experiment` and `/Outbound/experiment` on the PacketShaper, with host rules that matched the IP address of the FTP server (similar in principle to a `"host ftpserver"` tcpdump filter). I configured a rate policy of "38400 fixed" (38.4kpbs allotted bandwidth, no bursting beyond this rate) on these experiment classes. Then I placed the laptop on a network "Outside" of the PacketShaper, and repeated the FTP transfers. Here's what WinDUMP recorded:

```
22:24:08.225940 ftpserver.20 > laptop.1041: S 963321960:963321960(0)
win 24820 <mss 1380> (DF)
07:26:27.877701 laptop.1041 > ftpserver.20: S 9494537:9494537(0) ack
963321961 win 9324 <mss 1332> (DF)
22:24:08.228115 ftpserver.20 > laptop.1041: . ack 1 win 1380 (DF)
22:24:08.234390 ftpserver.20 > laptop.1041: P 1:1333(1332) ack 1 win
2760 (DF)
22:24:08.384244 laptop.1041 > ftpserver.20: . ack 1333 win 9324 (DF)
22:24:08.522543 ftpserver.20 > laptop.1041: P 1333:2665(1332) ack 1 win
2760 (DF)
22:24:08.683950 laptop.1041 > ftpserver.20: . ack 2665 win 9324 (DF)
22:24:08.808167 ftpserver.20 > laptop.1041: P 2665:3997(1332) ack 1 win
2760 (DF)
22:24:08.984830 laptop.1041 > ftpserver.20: . ack 3997 win 9324 (DF)
22:24:09.093952 ftpserver.20 > laptop.1041: P 3997:5329(1332) ack 1 win
2760 (DF)
22:24:09.284782 laptop.1041 > ftpserver.20: . ack 5329 win 9324 (DF)
22:24:09.379567 ftpserver.20 > laptop.1041: P 5329:6661(1332) ack 1 win
2760 (DF)
22:24:09.484759 laptop.1041 > ftpserver.20: . ack 6661 win 9324 (DF)
22:24:09.665250 ftpserver.20 > laptop.1041: P 6661:7993(1332) ack 1 win
2760 (DF)
22:24:09.784708 laptop.1041 > ftpserver.20: . ack 7993 win 9324 (DF)
```

```
22:24:09.947919 ftpserver.20 > laptop.1041: P 7993:8193(200) ack 1 win
2760 (DF)
22:24:09.950554 ftpserver.20 > laptop.1041: P 8193:9325(1132) ack 1 win
2760 (DF)
07:26:29.603767 laptop.1041 > ftpserver.20: . ack 9325 win 9324 (DF)
22:24:10.233569 ftpserver.20 > laptop.1041: P 9325:9525(200) ack 1 win
2760 (DF)
22:24:10.236124 ftpserver.20 > laptop.1041: P 9525:10657(1132) ack 1
win 2760 (DF)
07:26:29.889387 laptop.1041 > ftpserver.20: . ack 10657 win 9324 (DF)
...
22:24:15.966114 ftpserver.20 > laptop.1041: . 35433:36765(1332) ack 1
win 2760 (DF)
22:24:16.088746 laptop.1041 > ftpserver.20: . ack 36765 win 9324 (DF)
22:24:16.252627 ftpserver.20 > laptop.1041: . 36765:38097(1332) ack 1
win 2760 (DF)
22:24:16.388703 laptop.1041 > ftpserver.20: . ack 38097 win 9324 (DF)
22:24:16.537217 ftpserver.20 > laptop.1041: FP 38097:38797(700) ack 1
win 25308 (DF)
07:26:36.190752 laptop.1041 > ftpserver.20: . ack 38798 win 8624 (DF)
22:24:16.539367 laptop.1041 > ftpserver.20: F 1:1(0) ack 38798 win 8624
(DF)
22:24:16.540729 ftpserver.20 > laptop.1041: . ack 2 win 25308 (DF)
```

and tcpdump:

```
07:26:40.815999 ftpserver.20 > laptop.1041: S 3154602740:3154602740(0)
win 24820 <mss 1460> (DF)
07:26:40.817886 laptop.1041 > ftpserver.20: S 9494537:9494537(0) ack
3154602741 win 1332 <mss 1332> (DF)
07:26:40.818118 ftpserver.20 > laptop.1041: . ack 1 win 25308 (DF)
07:26:40.821023 ftpserver.20 > laptop.1041: P 1:1333(1332) ack 1 win
25308 (DF)
07:26:41.108866 laptop.1041 > ftpserver.20: . ack 1333 win 1332 (DF)
07:26:41.109153 ftpserver.20 > laptop.1041: P 1333:2665(1332) ack 1 win
25308 (DF)
07:26:41.394596 laptop.1041 > ftpserver.20: . ack 2665 win 1332 (DF)
07:26:41.394859 ftpserver.20 > laptop.1041: P 2665:3997(1332) ack 1 win
25308 (DF)
07:26:41.680356 laptop.1041 > ftpserver.20: . ack 3997 win 1332 (DF)
07:26:41.680629 ftpserver.20 > laptop.1041: P 3997:5329(1332) ack 1 win
25308 (DF)
07:26:41.965991 laptop.1041 > ftpserver.20: . ack 5329 win 1332 (DF)
07:26:41.966251 ftpserver.20 > laptop.1041: P 5329:6661(1332) ack 1 win
25308 (DF)
07:26:42.251717 laptop.1041 > ftpserver.20: . ack 6661 win 1332 (DF)
07:26:42.252007 ftpserver.20 > laptop.1041: P 6661:7993(1332) ack 1 win
25308 (DF)
07:26:42.537404 laptop.1041 > ftpserver.20: . ack 7993 win 1332 (DF)
07:26:42.537628 ftpserver.20 > laptop.1041: P 7993:8193(200) ack 1 win
25308 (DF)
07:26:42.537775 ftpserver.20 > laptop.1041: P 8193:9325(1132) ack 1 win
25308 (DF)
07:26:42.823099 laptop.1041 > ftpserver.20: . ack 9325 win 1332 (DF)
07:26:42.823322 ftpserver.20 > laptop.1041: P 9325:9525(200) ack 1 win
25308 (DF)
```

```
07:26:42.823483 ftpserver.20 > laptop.1041: P 9525:10657(1132) ack 1
win 25308 (DF)
07:26:43.108885 laptop.1041 > ftpserver.20: . ack 10657 win 1332 (DF)
...
07:26:48.553487 ftpserver.20 > laptop.1041: . 35433:36765(1332) ack 1
win 25308 (DF)
07:26:48.839538 laptop.1041 > ftpserver.20: . ack 36765 win 1332 (DF)
07:26:48.839852 ftpserver.20 > laptop.1041: . 36765:38097(1332) ack 1
win 25308 (DF)
07:26:49.125961 laptop.1041 > ftpserver.20: . ack 38097 win 1332 (DF)
07:26:49.126257 ftpserver.20 > laptop.1041: FP 38097:38797(700) ack 1
win 25308 (DF)
07:26:49.130594 laptop.1041 > ftpserver.20: . ack 38798 win 1332 (DF)
07:26:49.131555 laptop.1041 > ftpserver.20: F 1:1(0) ack 38798 win 8624
(DF)
07:26:49.131753 ftpserver.20 > laptop.1041: . ack 2 win 25308 (DF)
```

The PacketShaper starts its meddling with the very first SYN: it reduces the FTP server's Maximum-Segment-Size TCP option from 1460 to 1380.

The laptop always advertizes a window size of 9324, but the PacketShaper changes it to 1332 during the three-way handshake. Note that the reduced window is the same as the laptop's unchanged Maximum-Segment-Size option.

At the end of the three-way handshake, it also reduces the server's advertized window size from 24820 to 1380. Notice that value also matches the server's reduced Maximum-Segment-Size, and that it creeps the server's window size up to 2760 during the connection.

Also, the laptop's acknowledgements have been paced at about 286 milliseconds apart, so the PacketShaper hit the bandwidth target of 38400 bits per second pretty closely. Doing the math to calculate the resulting bandwidth:

> eight bits per byte × 1332 bytes per ack ÷ 286 ms per ack
> = 37258 bits per second.

When the FIN packets arrive, there's no more need to shape traffic, so those packets' window sizes are left unchanged in both directions.

The PacketShaper does a similar job for the FTP `PUT` file transfer. The laptop's WinDUMP sees:

```
22:24:23.129437 ftpserver.20 > laptop.1042: S 3870297829:3870297829(0)
win 24820 <mss 1380> (DF)
07:26:42.782890 laptop.1042 > ftpserver.20: S 9509443:9509443(0) ack
3870297830 win 9324 <mss 1332> (DF)
22:24:23.131606 ftpserver.20 > laptop.1042: . ack 1 win 1380 (DF)
22:24:23.148238 laptop.1042 > ftpserver.20: . 1:1333(1332) ack 1 win
9324 (DF)
22:24:23.434707 ftpserver.20 > laptop.1042: . ack 1333 win 1380 (DF)
07:26:43.089592 laptop.1042 > ftpserver.20: . 1333:2665(1332) ack 1 win
```

```
9324 (DF)
22:24:23.720577 ftpserver.20 > laptop.1042: . ack 2665 win 1380 (DF)
07:26:43.375523 laptop.1042 > ftpserver.20: . 2665:3997(1332) ack 1 win
9324 (DF)
22:24:24.006221 ftpserver.20 > laptop.1042: . ack 3997 win 1380 (DF)
07:26:43.661203 laptop.1042 > ftpserver.20: P 3997:5329(1332) ack 1 win
9324 (DF)
22:24:24.292030 ftpserver.20 > laptop.1042: . ack 5329 win 1380 (DF)
...
07:26:51.091385 laptop.1042 > ftpserver.20: P 38629:38797(168) ack 1
win 9324 (DF)
22:24:31.483052 ftpserver.20 > laptop.1042: . ack 38797 win 1380 (DF)
07:26:51.137428 laptop.1042 > ftpserver.20: F 38797:38797(0) ack 1 win
9324 (DF)
22:24:31.484934 ftpserver.20 > laptop.1042: . ack 38798 win 1380 (DF)
22:24:31.531139 ftpserver.20 > laptop.1042: F 1:1(0) ack 38798 win
25308 (DF)
07:26:51.185523 laptop.1042 > ftpserver.20: . ack 2 win 9324 (DF)
```

and the server's tcpdump:

```
07:26:55.720973 ftpserver.20 > laptop.1042: S 3156559273:3156559273(0)
win 24820 <mss 1460> (DF)
07:26:55.722934 laptop.1042 > ftpserver.20: S 9509443:9509443(0) ack
3156559274 win 1332 <mss 1332> (DF)
07:26:55.723170 ftpserver.20 > laptop.1042: . ack 1 win 25308 (DF)
07:26:55.743130 laptop.1042 > ftpserver.20: . 1:1333(1332) ack 1 win
2664 (DF)
07:26:55.743357 ftpserver.20 > laptop.1042: . ack 1333 win 25308 (DF)
07:26:56.031313 laptop.1042 > ftpserver.20: . 1333:2665(1332) ack 1 win
2664 (DF)
07:26:56.075427 ftpserver.20 > laptop.1042: . ack 2665 win 25308 (DF)
07:26:56.317302 laptop.1042 > ftpserver.20: . 2665:3997(1332) ack 1 win
2664 (DF)
07:26:56.365451 ftpserver.20 > laptop.1042: . ack 3997 win 25308 (DF)
07:26:56.602831 laptop.1042 > ftpserver.20: P 3997:5329(1332) ack 1 win
2664 (DF)
07:26:56.645400 ftpserver.20 > laptop.1042: . ack 5329 win 25308 (DF)
...
07:27:04.031205 laptop.1042 > ftpserver.20: P 38629:38797(168) ack 1
win 2664 (DF)
07:27:04.075392 ftpserver.20 > laptop.1042: . ack 38797 win 25308 (DF)
07:27:04.077082 laptop.1042 > ftpserver.20: F 38797:38797(0) ack 1 win
9324 (DF)
07:27:04.077310 ftpserver.20 > laptop.1042: . ack 38798 win 25308 (DF)
07:27:04.123526 ftpserver.20 > laptop.1042: F 1:1(0) ack 38798 win
25308 (DF)
07:27:04.125170 laptop.1042 > ftpserver.20: . ack 2 win 9324 (DF)
```

This time, WinDUMP sees the server's window reduced to 1380 throughout, and the tcpdump sees the laptop's window creep up to 2664. Again, the FIN packets are passed through unmolested.

I repeated these FTP `PUT` and `GET` experiments with different file sizes and different PacketShaper Rate policies, and found these behaviors repeated:

- During the three way handshake, the PacketShaper reduced window sizes to match Maximum-Segment-Sizes.
- The number of segments transmitted between acknowledgements decreased.
- The PacketShaper delayed and paced acknowledgement packets.
- FIN packets were transmitted unmolested.

So let's try to extrapolate from this experiment. What if we are studying some TCP traffic, and we suspect it has been managed by a PacketShaper, but we don't know for certain that such a device intervened? The following evidence might reveal the PacketShaper:

- Look for window sizes in FIN packets that are larger than the window sizes advertized throughout the connection they terminate.
- Look for evidence of pacing by examining timestamps between ACK packets, or counting the transmitted segments between ACKs.
- In a three-way handshake, look for a window size that matches an "mss" size.

The first property, large window sizes in FIN packets, is probably the most reliable tip-off. Note that since a PacketShaper changes initial TCP window sizes and sometimes interferes with the MSS TCP option in SYN packets, these values may not be reliable for use in passive OS fingerprinting analysis. (Miller; see also Passive Fingerprinting, Honeynet Project) However, I never noticed the PacketShaper tweak the window size of the *first* (SYN) packet in a three-packet handshake.

Other PacketShaper Tricks

Three PacketShaper features, packet capture, event notification, and graphical reporting, are illustrated here with real-world examples drawn from my own network security experiences.

Class Based Packet Capturing

Late in 2002 Packeteer added a **`packetlog`** command to the PacketShaper's *PacketWise* operating system software, version 5.3. It is used to log the packets matching a set of traffic classes to a libpcap (tcpdump format) file in the PacketShaper's internal storage, which must be transferred (via FTP) to a workstation for analysis. A troubleshooting tool intended to be used at the direction of Packeteer technical support professionals, it is also a boon to intrusion analysts.

Although it can log a very brief interval of all traffic by specifying that the root classes `/Inbound` and `/Outbound` be logged, It is more helpful when an event of interest is identified with traffic classes at leaves of the traffic tree. In particular, it gives an analyst an avenue for studying the traffic that the PacketShaper does not know how to classify and lumps into a default class such as `/Inbound/Default`. (Note: if `/Inbound/AutoDiscovered` is configured to log IP traffic, as explained above, then `/Inbound/Autodiscovered/Default` contains unclassified IP traffic and `/Inbound/Default` contains non-IP traffic.)

The following real-world example illustrates this feature.

For both inbound and outbound traffic, the PacketShaper I work with automatically created classes `AutoDiscovered/DiscoveredPorts/UDP_Port_7674`. Although there is a Sun Microsystems product that uses UDP port 7674, I suspected that this was actually traffic from a peer-to-peer application.

I used **`packetlog`** to capture a few thousand packets from this traffic class, and examined the traffic using ethereal. (My favorite tool, tcpdump, gives 'snaplen of 0 rejects all packets' errors on PacketShaper **`packetlog`** traces, so for these I use ethereal instead.)

The port 7674 traffic consisted of small, rapidly transmitted datagrams, with only about ten to thirty bytes of payload in each UDP packet. Most of the data looked like gibberish, but a very few packets happened to contain the names of some contemporary popular musicians and their song titles in readable ASCII.

I proposed applying the same policies to this class that we applied to the other peer-to-peer file trading applications; the **`packetlog`** output gave me circumstantial evidence to support that proposal.

PacketShaper Event Notification

The `event` commands may be used to define up to 32 *events*. An event is a condition that the PacketShaper can measure periodically, such as bandwidth used by a traffic class exceeding a certain rate, or traffic latency exceeding a certain time interval. When the condition becomes true, the event triggers an action, which may be either an SNMP trap, an email, or a syslog message.

Several times, a person unaffiliated with the university gained unauthorized access to one of our computing labs, and used a lab computer to upload objectionable material over the internet via HTTP. The PacketShaper's event notification feature was one of the tools we used to apprehend the trespasser. One of my colleagues created a subclass of `/Outbound/HTTP` that matched the perpetrator's upload URLs, applied a policy to control bandwidth on that subclass, and created an event to send email when uploads happened. The campus police peacefully took the uploader into custody.

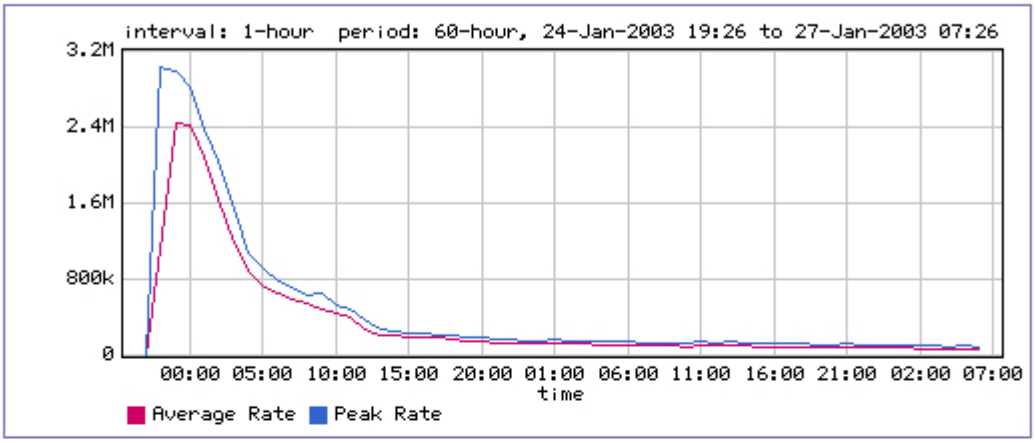
PacketShaper events are no substitute for a true intrusion detection system, but for a small number of high priority events of interest, they make excellent correlative data, especially if the PacketShaper is properly configured to synchronize its clock with an NTP server.

Charts and Graphs

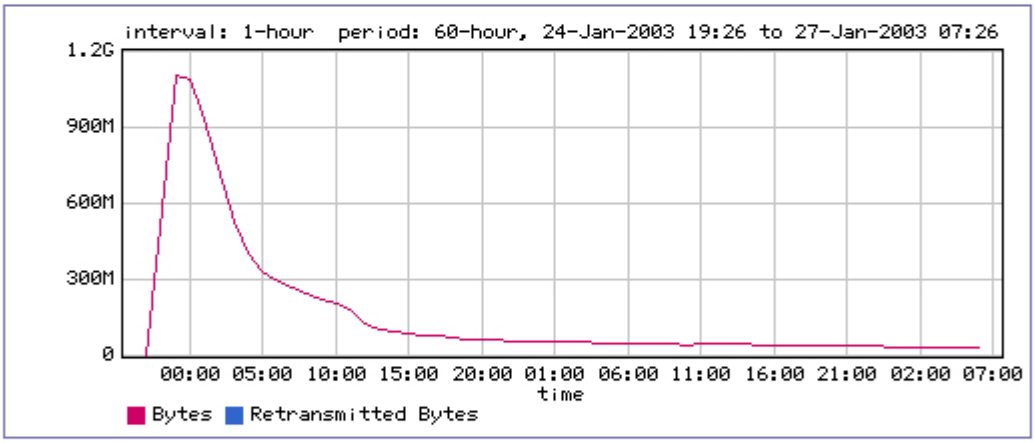
The reporting capabilities of the PacketShaper's web user interface are useful to intrusion analysts as well. For example, the "SQL Slammer" worm attacks on 25 January 2003 were automatically classified by our PacketShaper as /Inbound/AutoDiscovered/DiscoveredPorts/UDP_Port_1434. That made it easy to visualize the impact of the worm on our border traffic:

Bandwidth Utilization with Peaks Report
for Class /Inbound/AutoDiscovered/DiscoveredPorts/UDP_Port_1434

Class Utilization with Peaks



Bytes Transmitted



Packets Transmitted

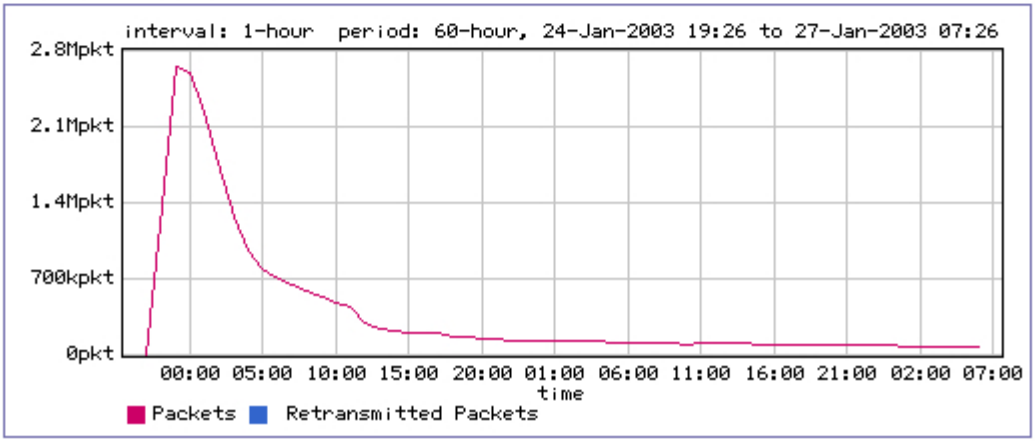


Figure: PacketShaper graphs of "SQL Slammer" border traffic

We noted from these graphs that the worm had never consumed more than about 3 Mbps of our inbound bandwidth at its peak, and at that peak more than 2,500,000 worm datagrams per second were handled by our border router. We verified that the router CPU handled that peak load without any strain, that the firewall was still blocking port 1434 traffic, and most importantly, that there was no *outbound* worm traffic on port 1434.

Satisfied that our network countermeasures were adequate, we turned our attention to more important matters, such as finding and patching potentially vulnerable machines inside the perimeter. We counted ourselves fortunate that we didn't contribute to the outbreak, nor suffer from it very much.

Stanford's Packeteer Listserv

I would like to acknowledge all who participate in the packeter-edu mailing list (SUNet) hosted by Stanford University, for their help and insights. A frequent topic on the list is management of peer-to-peer traffic, notably Kazaa (Sharman Networks), whose protocols change frequently, apparently for the purpose of defeating the PacketShaper's Traffic Classification and Shaping mechanisms.

References

Allot Communications. "Allot - bandwidth management, QoS, service level agreement, Internet filter." Home page. 2003. URL:http://www.allot.com/ (29 Jan. 2003)

Bell, Mansel. "Securing an Anonymous FTP Server in Solaris 8 with WU-FTPD." SANS Info Sec Reading Room. 30 Mar. 2002. URL:http://www.sans.org/rr/protocols/anonymous_ftp.php (29 Jan. 2003)

Cupps, John. "How to Identify and `Contain' Some of the Information Security Problems Created by Unique Business Environments." SANS Info Sec Reading Room. 10 Aug. 2001. URL:http://www.sans.org/rr/casestudies/infosec_problems.php (29 Jan. 2003)

Dawson, Terry. "Traffic Shaping." The O'Reilly Network. Linux Devcenter. 24 Aug. 2000. URL:http://linux.oreillynet.com/pub/a/linux/2000/08/24/LinuxAdmin.html (29 Jan. 2003)

Ethereal. "The Ethereal Network Analyzer." Home page. 25 Jan. 2003. URL:http://www.ethereal.com/ (29 Jan. 2003).

Disenchanted. "Tragedy of the commons." Disenchanted Dictionary. 2000-2002.

URL: http://www.disenchanted.com/dis/lookup?node=1218 (29 Jan. 2003)

Hardy, William C. QoS Measurement and Evaluation of Telecommunications Quality of Service. Chichester, England: John Wiley & Sons, Inc., 2001.

Honeynet Project. "Know Your Enemy: Passive Fingerprinting." 4 Mar. 2002. URL: http://project.honeynet.org/papers/finger/ (29 Jan. 2003)

IETF[1]. "Differentiated Services (diffserv) Charter." Internet Engineering Task Force Working Groups. 9 Sep. 2002. URL:http://www.ietf.org/html.charters/diffserv-charter.html (29 Jan. 2003)

IETF[2]. "Multiprotocol Label Switching (mpls) Charter." Internet Engineering Task Force Working Groups. 27 Jan. 2003. URL:http://www.ietf.org/html.charters/mpls-charter.html (29 Jan. 2003)

Jupitermedia Corporation. "Webopedia: The 7 Layers of the OSI Model." Webopedia. 2003. URL:http://www.webopedia.com/quick_ref/OSI_Layers.asp (29 Jan. 2003)

Kilkki, Kalevi. Differentiated Services for the Internet. Macmillan Technology Series. Indianapolis, IN: Macmillan Technical Publishing, 1999.

Lightspeed Systems. "Lightspeed Systems - Take Control of Your Network." Home page. 2003. URL:http://www.lightspeedsystems.com/ (29 Jan. 2003)

Miller, Toby. "Passive OS Fingerprinting: Details and Techniques." SANS Institute. 2001-2002. URL:http://www.incidents.org/papers/OSfingerprinting.php (29 Jan. 2003)

Nancarrow, Michael. "Protecting your Internal Systems from a Compromised Host." SANS Info Sec Reading Room. 26 Mar. 2002 URL:http://www.sans.org/rr/casestudies/host.php (29 Jan. 2003)

Packeteer[1]. "Packeteer Home." Home page. 2003. URL:http://www.packeteer.com/ (29 Jan. 2003)

Packeteer[2]. "TCP Rate Control and Alternatives." Packeteer White Paper Series. May 2002. URL:http://www.packeteer.com/solutions/resources/TcpRateControl.pdf (29 Jan. 2003)

Packeteer[3]. "UDP Traffic Management." Packeteer White Paper Series. May 2002. URL:http://www.packeteer.com/solutions/resources/UDPTrafficManagement.pdf (29 Jan. 2003)

RealNetworks, Inc. "RealSystem Firewall Support." RealNetworks Service and Support. URL:http://service.real.com/firewall/ (29 Jan. 2003)

Scarborough, Matt. "Re: windump timezone gotcha." E-mail, intrusions@incidents.org. 13 Nov 2001. URL:http://www.incidents.org/archives/intrusions/msg01736.html (29 Jan. 2003)

Sharman Networks. "Kazaa." Home Page. 2002. URL:http://www.kazaa.com/ (29 Jan. 2003).

Siegel, Eric D. Designing Quality of Service Solutions for the Enterprise. New York, NY: John Wiley & Sons, Inc., 2000.

Sitara Networks. "Sitara Networks: QoS solutions for enterprise and service provider markets." Home page. 2003. URL:http://www.sitaranetworks.com/ (29 Jan. 2003)

Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Addison Wesley Professional Computing Series. Reading, MA: Addison Wesley Longman, Inc., 1994.

Stubbs, Robin. "windump timezone gotcha." E-mail, intrusions@incidents.org. 7 Nov 2001. URL:http://www.incidents.org/archives/intrusions/msg01681.html (29 Jan. 2003)

SUNet. "SUNet Systems Networking Lists Packeteer Archives." Stanford University. 2001-2002. URL: http://www.stanford.edu/group/networking/netlists/ (29 Jan. 2003)

Part 2

Network Detects

NETWORK DETECT 1:

A compromised internal host attempts to flood an external host with IP protocol 255 packets.

1. Source of Trace:

This occurred at my place of employment (a university campus) in August 2002.

The network topology looks like this:

```
faculty host ----- Switch ----- Switch ----- Campus
Firewall
                                ^
                          (backbone link)
```

where the backbone link is Gigabit Ethernet, and all others are FastEthernet.

2. Detect Generated by:

A colleague found a switch interface carrying traffic far exceeding expected volume. He saw this occur more than once, each time for intervals
lasting several minutes. We identified the hosts behind that switch interface,
and began to monitor traffic; I saved the monitored traffic
in a tcpdump binary file. At that time, we had not yet deployed an intrusion
detection system.

The tcpdump log
revealed a steady stream of outgoing packets:

```
13:07:09.303325 0:d0:b7:90:1a:98 0:50:b:6a:2c:0 0800 1474:

 xxx.xxx.192.249

> 211.13.169.33:  ip-proto-255 1440

 13:07:09.303567 0:d0:b7:90:1a:98 0:50:b:6a:2c:0 0800 1474:

 xxx.xxx.192.249

> 211.13.169.33:  ip-proto-255 1440

 13:07:09.303687 0:d0:b7:90:1a:98 0:50:b:6a:2c:0 0800 1474:

 xxx.xxx.192.249
```

```
> 211.13.169.33:  ip-proto-255 1440

 13:07:09.303807 0:d0:b7:90:1a:98 0:50:b:6a:2c:0 0800 1474:

 xxx.xxx.192.249

> 211.13.169.33:  ip-proto-255 1440

 13:07:09.303934 0:d0:b7:90:1a:98 0:50:b:6a:2c:0 0800 1474:

 xxx.xxx.192.249

> 211.13.169.33:  ip-proto-255 1440

 13:07:09.304046 0:d0:b7:90:1a:98 0:50:b:6a:2c:0 0800 1474:

 xxx.xxx.192.249

> 211.13.169.33:  ip-proto-255 1440

 13:07:09.304167 0:d0:b7:90:1a:98 0:50:b:6a:2c:0 0800 1474:

 xxx.xxx.192.249

> 211.13.169.33:  ip-proto-255 1440

 13:07:09.304286 0:d0:b7:90:1a:98 0:50:b:6a:2c:0 0800 1474:

 xxx.xxx.192.249

> 211.13.169.33:  ip-proto-255 1440

 13:07:09.304407 0:d0:b7:90:1a:98 0:50:b:6a:2c:0 0800 1474:

 xxx.xxx.192.249

> 211.13.169.33:  ip-proto-255 1440

 ...
```

I used tcpdump's **-X** option to
find that the payloads were almost, but not quite, uniform.
(Unfortunately, I failed to increase the default snap length

when I obtained the tcpdump binary logfile, so I cannot share the entire payloads.)

```
13:07:09.303325 xxx.xxx.192.249 > 211.13.169.33:  ip-proto-255 1440

0x0000   4500 05b4 0e09 0000 fdff xxxx xxxx c0f9

E..............

0x0010   d30d a921 aaaa aaaa aaaa aaaa aaaa aaaa

...!..........

0x0020   aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa

................

0x0030   aaaa aaaa aaaa aaaa 60dc aaaa aaaa aaaa

........`.......

0x0040   aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa

................

0x0050   aaaa

..

13:07:09.303567 xxx.xxx.192.249 > 211.13.169.33:  ip-proto-255 1440

0x0000   4500 05b4 0061 0000 fdff xxxx xxxx c0f9

E....a.........

0x0010   d30d a921 aaaa aaaa aaaa aaaa aaaa aaaa

...!..........

0x0020   aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa

................
```

```
0x0030   aaaa aaaa aaaa aaaa 01a5 aaaa aaaa aaaa

.................

0x0040   aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa

.................

0x0050   aaaa

..


...
```

Measurements of traffic at our WAN link
confirm that this traffic did not escape through
the border router, so the attempted DoS was unsuccessful. However,
we did not detect any ICMP protocol unreachable messages from the
border
router.

3. Probability that the source address was spoofed:

None; we identified the attacking host.

4: Description of attack:

This is "out of spec" (OOS) traffic,
and it represents an attempted denial-of-service
(DoS) attack against cdn01.cdn.apinetland.net.

In every IPv4 packet, the byte at offset 9
identifies the protocol encapsulated
in the packet. The most common protocl numbers are 1 (ICMP), 6 (TCP),
and 17 (UDP). But this traffic has a protocol number of 255. In
the document "Protocol Numbers" at
http://www.iana.org/assignments/protocol-numbers,
the Internet Assigned Numbers Authority (IANA) lists assignments and
technical references
for protocol numbers 0 through 134. Protocols 135 through 254 are
"unassigned,"

and 255, the highest possible protocol number, is identified as "reserved."

These packets have large payloads (1440 bytes); each packet uses slightly
less than Ethernet's MTU. And they are sent very rapidly; the above trace
shows 8 packets detected in a one-microsecond span. Some arithmetic: eight
bits per byte times 1440 bytes per packet times 8000 packets per second
reckons at 92,160,000 bits per second; add overhead and it appears that
the attack completely saturated the 100Mbps of
bandwidth available to the host.

5. Attack mechanism

These are crafted packets. Tools are freely
available for packet crafting and injection.
For example, hping2 (http://www.hping.org) (hping)
can be made to create protocol
255 traffic, using a command such as
hping2 --rawip --ipproto 255 $target;
there are other options for specifying data payload size and injection interval.
Packet injection tools may also be created from scratch using a C compiler.

During incident handling and forensics on the attacking host, we found logfile
evidence indicating a remote compromise.
However, we did not find any rootkits, nor did we
find the tool used to craft these packets.

6. Correlations:

It's possible that this was part of a distributed denial-of-service against
the target, or in other words, that hosts at other sites
were attempting the same attack concurrently.

Alefiya Hussain and others at the University of Southern California presented
a discussion entitled "A Method for Differentiating DDoS Attacks" as part

of a Computer Communications course (http://netweb.usc.edu/cs551/) (Papadopoulos), in which
they identify a distributed denial-of-service against a .usc.edu machine
by 28 attackers using protocol 255 packets. At the time of this writing,
the Microsoft Power Point document of their presentation is available at
http://netweb.usc.edu/cs551/slides/alefiya.ppt. (Hussain)

Crist J. Clark also noted some IP protocol 255 traffic in the wild, and posted
an email about it on 11 July 2002 (archived at
http://lists.jammed.com/incidents/2002/07/0068.html) (Clark).
But the traffic he saw had a different payload; he found ICMP traffic
embedded in the payload of the protocol 255 traffic.

7. Evidence of active targeting:

This was not a random scan, and no stealth was attempted; this was a very
noisy attack aimed at a specific target.

DNS identifies the destination host 211.13.169.33
as "cdn01.cdn.apinetland.net," and
http://www.apnic.net/apnic-bin/whois2.pl (APNIC Whois) identifies this
as a Hitachi netBusiness Ltd. address.

8 Severity:

Criticality: **(3)**
The compromised host was a server used by a former
faculty member in his research work. The faculty member was no longer
with the university, and the host no longer served a critical function. However,
he had requested and obtained a hole in the campus firewall so that off-campus
clients could access it, and when he left the university, the breach was
not closed. Ability to access the host through a firewall increases
its otherwise low criticality score.

Lethality: **(5)** The attack consumed all of its local bandwidth and a
significant amount of the bandwidth on one of the backbone links. Were

it not for network countermeasures, the attack would also have consumed all
of the expensive bandwidth at the WAN link, and probably would have dissipated
bandwidth at the intended target as well.

System Countermeasures: **(1)** The compromised host remained online and neglected for a considerable period.

Network Countermeasures: **(3)** The border router prevented the attack, but an out-of-date firewall configuration allowed the compromise that enabled
the attack attempt.

Criticality + Lethality - System Countermeasures - Network Countermeasures
= 4; this event was worthy of our time and effort.

9: Defensive recommendation:

The campus firewall configuration and policies should be regularly audited.
For every "hole" poked through the firewall for a server, a person responsible
for maintaining that server should be contacted regularly to discuss its
potential vulnerabilities. The border router configuration regarding ICMP
protocol unreachable packets should also be reviewed; ICMP protocol unreachable
packets may be a desirable diagnostic feature, but we don't desire they be
abused by attackers for protocol mapping reconnaissance. (It appears this
attack did not produce any protocol unreachable messages.)

10: Multiple choice test question:

An IP protocol number of 255 represents what kind of traffic?

a) Packets of other IP protocols, such as ICMP or TCP, are embedded in the payload of protocol 255 packets.

b) Datagrams of IPv6 protocols transmitted via IPv4 must be embedded in the payload of protocol 255 packets.

c) Distributed Denial-of-Service traffic.

d) Protocol ID 255 is reserved by the Internet Assigned Numbers Authority
(IANA); so any such traffic is not valid.

answer: d.

References

APNIC Whois. "Query the APNIC Whois Database."
Asia Pacific Network Information Centre.
URL:http://www.apnic.net/apnic-bin/whois2.pl (29 Jan. 2003)

Clark, Crist J. "Protocol 255." E-mail to SecurityFocus Online INCIDENTS
mailing list, archived at lists.jammed.com. 11 Jul. 2002.
URL:http://lists.jammed.com/incidents/2002/07/0068.html (29 Jan. 2003)

hping. Home Page.
URL:http://www.hping.org/ (29 Jan. 2003)

Hussain, Alefiya, et. al. "A Method for Differentiating DDoS Attacks."
Microsoft PowerPoint presentation document,
course material, University of Southern California. Fall 2002.
URL:http://netweb.usc.edu/cs551/slides/alefiya.ppt (29 Jan. 2003)

IANA. "Protocol Numbers." Internet Assigned Numbers Authority.
URL:http://www.iana.org/assignments/protocol-numbers (29 Jan. 2003)

Papadopoulos, Christos.
"Computer Communications." Course Material, University of
Southern California. Fall 2002.
URL:http://netweb.usc.edu/cs551/ (29 Jan. 2003)

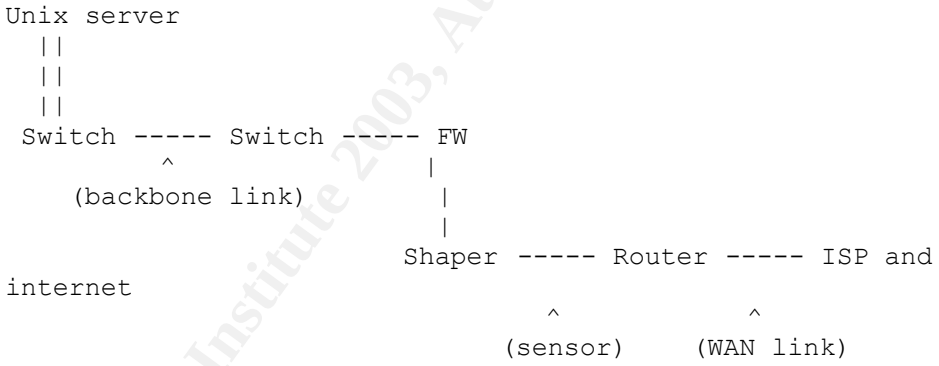
NETWORK DETECT 2:

A UDP flood from a compromised host shuts down the WAN link. Subsequent incident handling finds the code responsible for the attack.

1. Source of Trace:

This occurred at my place of employment (a university campus) in September 2002.

The network topology looks like this:



(*Legend*: The Unix host sports two Gigabit Ethernet interfaces. The backbone link consists of two 1000BaseX fiber lines, for a total bandwidth of 2000Mbps. The WAN link is a 30Mbps portion of a 155Mbps ATM circuit. The rest of the links are 100Mbps Fast Ethernet. The "FW" is our firewall, and the "Shaper" is a *PacketShaper*® traffic management device from Packeteer®, Inc. (Packeteer) It employs technology to classify each traffic flow, and shape traffic so that our mission cricital applications don't have their bandwidth depleted by lower priority flows.)

Here is a narrative of the incident:

A month had passed since the *SANS Beyond Firewalls* conference and training week, and I was busy getting my "newbie" feet wet, trying to set up a SHADOW NIDS (SHADOW). During an evening of working from home, my terminal sessions became unresponsive (up to a minute between typing a character and seeing it echo back to me on the screen).

The University's ISP provides a diagnostic web site with traffic graphs (similar to the popular MRTG graphs) (Oetiker and Rand). I checked and found that our outbound traffic rate at the WAN link was pegged at 30Mbps, our maximum. It stayed that way for about ten minutes. I telephoned and alerted my network operations colleagues.

When the WAN link was usable again, I quickly started a **tcpdump -w $logfile -i $sensor_interface** process, to sniff all of the campus border link traffic in case it happened again. It happened again. The second attack succeeded in overwhelming and crashing the campus firewall, effectively closing the WAN link completely for a short time (a second firewall was configured in an "automatic failover" configuration, and the second unit took about a minute to activate).

Meanwhile, the network gurus traced this second traffic tsunami to a particular host. It was our main Unix server, the most powerful machine on campus: four very fast processors, two 1000BaseX network interfaces, many software packages (including compilers) installed, and tens of thousands of student, faculty, and staff shell accounts.

The system administrator on duty identified the offending process (it was consuming most of the CPU cycles on one of the four processors) and its owner's UID. That account's files were made available to the network security team. Syslog evidence indicated that the account was being used by somebody other than its owner.

The tcpdump log revealed a steady stream of outgoing packets.
(Regretfully, I neglected to increase tcpdump's default snap length.)

```
21:03:48.207061 attacker.4246 > victim.53:  12594 op6+$
[b2&3=0x3334] [14136a] [13622q] [14640n] [49au][|domain]
(DF)


0x0000  4500 004d ae33 4000 1c11 a37c atta cker
E..M.3@....|....


0x0010  .vic tim. 1096 0035 0039 6fbb 3132 3334
.......5.9o.1234


0x0020  3536 3738 3930 0031 3200 3132 3300 3132
567890.12.123.12


0x0030  3334 0031 3233 3435 0031 3233 3435 3600
34.12345.123456.


0x0040  3132 3334 3536 3700 3132 3334 35
1234567.12345


21:03:48.207176 attacker.4246 > victim.53:  12594 op6+$
[b2&3=0x3334] [14136a] [13622q] [14640n] [49au][|domain]
(DF)


0x0000  4500 004d ae38 4000 1c11 a377 atta cker
E..M.8@....w....


0x0010  .vic tim. 1096 0035 0039 6fbb 3132 3334
.......5.9o.1234


0x0020  3536 3738 3930 0031 3200 3132 3300 3132
567890.12.123.12


0x0030  3334 0031 3233 3435 0031 3233 3435 3600
34.12345.123456.
```

```
0x0040  3132 3334 3536 3700 3132 3334 35
1234567.12345
```

...and many many thousand more packets like these. (The source and destination IP addresses have been anonymized; I cannot divulge either at this time. An IRC server operated at the off campus destination IP address.)

2. Detect Generated by:

Tcpdump, version 3.6.2.

3. Probability source address was spoofed:

None. In the narrative of the attack, the source address in the packets is censored, but that address was not spoofed. Furthermore, egress filtering is configured on the border router to prevent packets spoofing non-DU addresses from escaping.

4: Description of attack:

This was a DNS UDP flood attack, and it represents a successful denial-of-service (DoS) attack.

Prior to the attack, the perpetrator gained unauthorized access to an account on the Unix host, and downloaded and compiled the exploit code. We are of the opinion that the perpetrator stole, guessed, or cracked the account's password.

5. Attack mechanism

These are crafted packets, sent as rapidly as possible to waste bandwidth.
Since they use UDP port 53, the same as DNS, the PacketShaper classified
this traffic as mission critical, so the attacker succeeded in wasting all
of our WAN link bandwidth.

6. Correlations:

The compromised account contained a subdirectory named " /" (space) containing two files. One was the attacker's executable, and the other was
the C source file the attacker used to build the executable.
Using the name of the source file as a search key,
Google
found sites
offering downloads of the exploit code, and mailing list
archives containing messages from victims of this exploit.

The following comment and code excerpts are from the C source file.
It compares well with the sample exploit code demonstrated in
the *SANS Intrusion Detection In-Depth* course materials.
(*Note:*
remember that SANS and
GIAC want to serve security professionals,
not the "black hat" community that we combat. Although publishing
the entire code here would be educational,
I have decided to err on the side of confidentiality.
In the spirit of
our Code of Ethics (GIAC), I have
included here only short excerpts that correlate with this detect, and
avoided publishing complete working exploit code. I have also removed
the author's names and aliases from the code.)

(Excerpt 1) This C comment snippet brings to mind the stereotypes many of us imagine about black hat community members. The author modestly claims
it would deplete 70% of our bandwidth; it did worse.

```
/*
############################################################
##################

  ...[snipped]...

   # It's better than stealth or nestea , teardrop or
something else ..
trust me .. i know that .     #
   # Sorry , but if u wanna spoof the adress
se only 127.0.0.1 , so ..... it's the only one who work. #
   # , and this program use 90% of CPU, and 70 % of ur
Band.... that's all

  ...[snipped]...

*/
```

(Excerpt 2) In our case, the source address was not spoofed, but it looks like the tool's author wanted to provide that capability:

```
   printf("Kick your ass %s,with flood on port %d spoofed
as %s\n", server,
port, spoof);
   printf("Flooding ... \n");
   hp = gethostbyname(server);
   if (hp==NULL) {
      printf("Unknown host: %s\n",server);
      exit(0);
   }
```

(Excerpt 3): More network code. SOCK_DGRAM correlates with the UDP protocol in the attack.

```
   thesock = socket(AF_INET, SOCK_DGRAM, 0);
```

(Excerpt 4): part of the code used to craft datagram payload:

```
   switch (c1)
                {
                 case '1' : flood_STRING="1234567890"
; break ;
                 case '2' : flood_STRING="12"; break ;
                 case '3' : flood_STRING="123"; break ;
                 case '4' : flood_STRING="1234"; break ;
                 case '5' : flood_STRING="12345"; break ;
                 case '6' : flood_STRING="123456"; break ;
                 case '7' : flood_STRING="1234567"; break ;
```

```
                    case '8' : flood_STRING="12345678"; break
;
                    case '9' : flood_STRING="123456789"; break
;
                    default  : flood_STRING="1234567890" ;
break ;
                  }
```

(Excerpt 5): An unterminated loop for sending the packets out as fast as possible:

```
                          for(;;)
                          {
                                  send(s, flood_STRING,
flood_SIZE, 0);
                           }
```

7. Evidence of active targeting:

All packets in the flood shared the same source and destination IP address.
The perpetrator was either trying to disrupt our service or disrupt an IRC
server at the destination address. I believe both.

8 Severity:

Criticality: **(5)** The compromised host was a high profile prize: lots of CPU, bandwidth, and shell accounts with access to compilers.

Lethality: **(5)** The attack consumed much of its local bandwidth, a good deal of the backbone link bandwidth, and all of the WAN link bandwidth (until it crashed the firewall). Furthermore, our WAN link actually has more capacity than we pay for, and we are expected to police our own traffic and keep our bandwidth usage under our contracted cap. If the attack had exceeded our limit for an extended period and cased a breach of contract, we could have been liable for hefty overcharges and suffered significant financial harm.

System Countermeasures: **(2)**

Since it's such an important system, qualified
personnel were available on call during the incident, and they
quickly identified the offending account and stopped the attack,
without
undue loss of service to the computer's other concurrent users.
Nevertheless, tens of thousands of shell accounts represent a
password compromise waiting to happen, and even though the system
was hardened (patches up to date, tcp_wrappers, quotas, setuid
programs disabled, etc.), none of those
efforts stop users from divulging a password or using a weak one.
Installed compilers exacerbate the risk posed by unauthorized account
access.

Network Countermeasures: **(3)**
The PacketShaper
successfully kept our WAN link bandwidth usage within contract limits,
but since DNS traffic was configured among traffic with highest
priority,
the DNS UDP flood was allowed to consume all of our contracted
WAN bandwidth. Also, we experienced about a minute of
complete downtime between the primary firewall's failure
and the second unit's subsequent activation.

Criticality + Lethality - System Countermeasures - Network
Countermeasures
= 5; a serious and sobering event.

9: Defensive recommendations:

-
- Only grant shell access to users that need it.
- Only grant compiler access to shell users that need it.
-
-
- Conduct an education campaign
- regarding the importance of choosing strong passwords
- and keeping them secret.
-
-
- Measure "normal" DNS traffic bandwidth, decide on a reasonable
- upper bound for DNS bandwidth,
- and configure the PacketShaper to enforce this limit.
-
-
- One question that remains unanswered: was this flood responsible

- for the firewall's failure, or was there some other attack used
- in tandem that brought down the firewall? Occam's Razor chooses
- the former, so continue the current automatic failover firewall
- configuration.
-

10: Multiple choice test question:

Consider the following (C language) statement from exploit source code:

```
thesock = socket(AF_INET, SOCK_DGRAM, 0);
```

and the following excerpt from a Unix system's "manpage" documentation for the socket() function:

```
SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       int socket(int domain, int type, int protocol);

DESCRIPTION
       Socket creates an endpoint for communication and
returns a
       descriptor.
...
       The  socket  has  the  indicated type, which
specifies the
       communication semantics.  Currently defined types
are:

       SOCK_STREAM
              Provides sequenced, reliable, two-way,
connection-
              based  byte streams.  An out-of-band data
transmis-
              sion mechanism may be supported.

       SOCK_DGRAM
              Supports datagrams (connectionless,
unreliable mes-
              sages of a fixed maximum length).
```

What protocol is most likely used in packets generated by the exploit code?

a) TCP

b) UDP

c) HTTP

d) SOCKS

answer: b.

References

GIAC. "GIAC Code of Ethics." Global Information Assurance Certification,
Advisory Board. URL:http://www.giac.org/COE.php (29 Jan. 2003)

Oetiker, Tobias and Rand, Dave. "MRTG Index Page." Sample graphs, Multi Router Traffic Grapher.
URL:http://www.stat.ee.ethz.ch/mrtg/ (29 Jan. 2003)

Packeteer[1]. "Packeteer Home." Home page. 2003.
URL:http://www.packeteer.com/ (29 Jan. 2003)

SHADOW. "NSWC SHADOW Index." SHADOW Home Page.
URL:http://www.nswc.navy.mil/ISSEC/CID/ (29 Jan. 2003)

NETWORK DETECT 3:

A spammer seeking an exploitable mail relay triggers

a pair of "WEB-CGI formmail" alerts.

1. Source of Trace:

http://www.incidents.org/logs/Raw/2002.10.15,
a sanitized tcpdump binary logfile
generated by an undisclosed Snort rule set.
Details of the logfile's sanitization are explained at
http://www.incidents.org/logs/Raw/README.
These changes include
modified IP addresses of the protected network,
modified checksums, and in some cases,
censored payloads. We'll see one such payload in this detect.

By viewing the logfile using
tcpdump -en -r 2002.10.15,
we can infer a few facts about the network in question.
The Snort sensor appears
to sit at the border of the 170.129.x.x
"class B" network space, because all traffic is between
ethernet MAC addresses 00:00:0c:04:b2:33
(the router interface for the class B, or "private" hosts)
and 00:03:e3:d9:26:c0
(the router interface for Internet, or "public" hosts). We'll assume that
170.129.x.x addresses have been sanitized.

2. Detect Generated by:

StillSecure Border Guard (Latis)
version 3.2.1,
a commercial NIDS/NIPS based on
Snort. (Snort)
The command was
/usr/local/stillsecure/snort/snort
-c snort.conf -l snort.logs -r 2002.10.15,
with the following key variables set in snort.conf:

```
var HOME_NET 170.129.0.0/16

var EXTERNAL_NET !$HOME_NET

var HTTP_SERVERS $HOME_NET

var HTTP_PORTS 80 443
```

The alert instances we shall consider are:

```
[**] [1:884:8] WEB-CGI formmail access [**]

[Classification: access to a potentially vulnerable web
application]

[Priority: 2]

11/14-18:09:02.206507 4.63.172.188:2394 ->
170.129.50.3:80

TCP TTL:109 TOS:0x0 ID:54336 IpLen:20 DgmLen:347 DF

***AP*** Seq: 0x8CE79981  Ack: 0x8EB8E04C  Win: 0x3A98

TcpLen: 20

[Xref => arachnids 226][Xref => cve CVE-1999-0172][Xref
=> bugtraq

1187][Xref => nessus 10076][Xref => nessus 10782]
```

```
[**] [1:884:8] WEB-CGI formmail access [**]
```

```
  [Classification: access to a potentially vulnerable web
application]

  [Priority: 2]


  11/14-18:21:41.316507 4.63.172.188:3624 ->
170.129.50.3:80


  TCP TTL:109 TOS:0x0 ID:51292 IpLen:20 DgmLen:347 DF


  ***AP*** Seq: 0xB1A1867E  Ack: 0xBDD0137B  Win: 0x3A98

  TcpLen:  20


  [Xref => arachnids 226][Xref => cve CVE-1999-0172][Xref
=> bugtraq

  1187][Xref => nessus 10076][Xref => nessus 10782]
```

and the Snort rule that triggered these alerts is:

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

Assuming the relevant snort.conf variables are correct, this rule produces a "WEB-CGI formmail access" alert when web traffic contains the string "/formmail" in a URI. The rule is documented with five references:

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

We shall use some of these in the analysis that follows.

In this detection instance, the source ephemeral port is 2394 and the source address is 4.63.172.188,
which DNS identifies as a Verizon DSL customer,
"tamqfl1-ar2-4-63-172-188.tamqfl1.dsl-verizon.net."
We may verify the "/formmail" string in
the logged packets' payloads using tcpdump's -X option.
Their TCP payloads are identical:

```
18:09:02.206507 4.63.172.188.2394 > 170.129.50.3.http: P
 2363988353:2363988660(307) ack 2394480716 win 15000 (DF)
  0x0000   4500 015b d440 4000 6d06 aadc 043f acbc
 E..[.@@.m....?..
  0x0010   aa81 3203 095a 0050 8ce7 9981 8eb8 e04c
 ..2..Z.P.......L
  0x0020   5018 3a98 df0b 0000 4745 5420 2f63 6769
```

```
P.:.....GET./cgi

 0x0030   2d62 696e 2f66 6f72 6d6d 6169 6c2e 706c
-bin/formmail.pl

 0x0040   3f65 6d61 696c 3d66 3240 616f 6c2e 636f
?email=f2@aol.co

 0x0050   6d26 7375 626a 6563 743d 7777 772e 5858
m&subject=www.XX

 0x0060   5858 5858 5858 2f63 6769 2d62 696e 2f66
XXXXXX/cgi-bin/f

 0x0070   6f72 6d6d 6169 6c2e 706c 2672 6563 6970
ormmail.pl&recip

 0x0080   6965 6e74 3d73 656e 6469 7078 7831 4079
ient=sendipxx1@y

 0x0090   6168 6f6f 2e63 6f6d 266d 7367 3d77 3030
ahoo.com&msg=w00

 0x00a0   7420 6168 6f6f 2532 4563 6f6d 266d 7367
t.ahoo%2Ecom&msg

 0x00b0   3d77 3030 7420 4854 5450 2f31 2e31 436f
=w00t.HTTP/1.1Co

 0x00c0   6e74 656e 742d 5479 7065 3a20 6170 706c
ntent-Type:.appl

 0x00d0   6963 6174 696f 6e2f 782d 7777 772d 666f
ication/x-www-fo

 0x00e0   726d 2d75 726c 656e 636f 6465 640d 0a55
```

```
rm-urlencoded..U

 0x00f0   7365 722d 4167 656e 743a 2047 6f7a 696c

ser-Agent:.Gozil

 0x0100   6c61 2f34 2e30 2028 636f 6d70 6174 6962

la/4.0.(compatib

 0x0110   6c65 3b20 4d53 4945 2035 2e35 3b20 7769

le;.MSIE.5.5;.wi

 0x0120   6e64 6f77 7320 3230 3030 290d 0a48 6f73

ndows.2000)..Hos

 0x0130   743a 2077 7777 2e58 5858 5858 5858 580d

t:.www.XXXXXXXX.

 0x0140   0a43 6f6e 6e65 6374 696f 6e3a 204b 6565

.Connection:.Kee

 0x0150   702d 416c 6976 650d 0a0d 0a

p-Alive....
```

The datagrams feature an HTTP GET request. The packet payload has been modified to anonymize a host name; here it appears as www.XXXXXXXX. The request may be easier to study when decoded and formatted in plain-text:

```
GET /cgi-
bin/formmail.pl?email=f2@aol.com&subject=www.XXXXX\

    XXX/cgi-
bin/formmail.pl&recipient=sendipxx1@yahoo.com&m\
```

```
    sg=w00t ahoo.com&msg=w00t HTTP/1.1Content-
Type: applica\


    tion/x-www-form-urlencoded



User-
Agent: Gozilla/4.0.(compatible; MSIE 5.5; windows 2000)



Host: www.XXXXXXXX



Connection: Keep-Alive
```

RFC 2068 (IETF)
describes the HTTP/1.1 protocol.
This HTTP GET request has fascinating traits and irregularities:

-
- At offset 0xef we see an uncommon client identification in the header:
- "User-Agent: Gozilla/4.0 (compatible; MSIE 5.5; windows 2000)."
- Googling
- for Gozilla/4.0 finds it in many publicly accessible web-usage logs and summaries, so it does seem to be widespread in the wild.
- Google also finds this "Gozilla" in web access log
- entries that Waldo Kitty posted to a SpamCop mailman list, http://news.spamcop.net/pipermail/spamcop-list/2002-April/000254.html
- (Kitty),
- and others H D Moore posted to a "Mobile Code" mailing list,
- http://citadelle.intrinsec.com/mailing/current/HTML/ml_mobile_code/0487.html (Moore).
- Those log entries bear a striking resemblance to this detect, so we'll
- correlate them in the forthcoming analysis.

Gozilla.com (DigitalCandle) sells a product called *Go!zilla*, an HTTP client that downloads many web pages at once. I obtained the free download of their version 4.11 and analyzed its traffic; it produces "Mozilla" User-Agent headers. I did not see it produce any "Gozilla" strings like the ones in this detect. (I cannot easily verify whether an earlier version is responsible for this traffic. My gut feeling is that *Go!zilla* is not guilty.)

A properly formed request URI terminates with the string HTTP/1.1, followed by a CRLF sequence (carriage return and line feed, hex 0d 0a) and then request header information strings, if any. In this payload, look at offset 0xbe. There is no "0d 0a" between HTTP/1.1 and the first request header, "Content-Type: application/x-www-form-urlencoded." The logs in the SpamCop and digitaloffense list e-mails referenced above have the same "missing 0d 0a" anomaly; more about this in the "Correlations" section below. Buggy HTTP client code probably causes this.

The string "w00t" (a "Leet-speak" spelling for "root") at offset 0x9d raises a red flag, and the e-mail address f2@aol.com at offset 0x47 is interesting too. These appear in Kitty's and Moore's posts as well.

- To see another irregularity,
- look at offsets 0x90 and 0xa2; we find the same 17-character substring
- "ahoo.com&msg=w00t" repeated at these offsets.
- (The dot is expanded as its hex code "%2E" in the second substring.)
- This too could be output from buggy HTTP code,
- but I prefer another explanation.

-

- I believe this
- anomaly is an artifact of the packet content anonymization algorithm
- applied to the 2002.10.15 logfile.
- I suspect that XXXXXXXX represents more than eight censored characters.
- The rest of the URI was decoded
- (%2E sequences replaced with dots) and shifted.
- But the entire payload contents
- were not shifted, just the URI,
- leaving the leftover "ahoo%2Ecom&msg=w00t" extant
- in the censored packet.
-

-

- (Side note: my paranoid brain also wonders if
- the GIAC packet sanitizers take sadistic pleasure in
- torturing GCIA candidates with such payload alterations.
- If the sanitizer is also the grader evaluating this practical detect,
- please accept my sincerest apology. <grin!>)

-

3. Probability that the source address was spoofed:

Unlikely. A three-way TCP handshake was completed prior the client sending this TCP datagram. In the upcoming description of the attack we'll
also see that the attacker was expecting a useful response to this HTTP GET.
The arachnids reference at http://www.whitehats.com/info/IDS226/ also mentions these two facts and agrees that the source address was not spoofed.

4: Description of attack:

The perpetrator seeks a web server offering Matt Wright's FormMail (http://www.scriptarchive.com/formmail.html) (Wright), a CGI script (formmail.pl) web-form to e-mail gateway. The attacker wants to exploit an input validation vulnerability to make FormMail send to any e-mail address. The script then becomes a vehicle for spamming (sending unsolicited bulk e-mail).

The references in the Snort rule mention other potential vulnerabilities as well. The Nessus reference at http://cgi.nessus.org/plugins/dump.php3?id=10782 identifies this "spamming" vulnerability, but also warns of "file disclosure, environment variable disclosure, and more." The other Nessus reference, http://cgi.nessus.org/plugins/dump.php3?id=10076, is an older warning; earlier FormMail versions' vulnerabilities allowed attackers to execute arbitrary code on the web server. The bugtraq reference at http://online.securityfocus.com/bid/1187 discusses the disclosure vulnerabilities, and also lists several patches available to apply to older versions of FormMail.

Nevertheless, the packet data suggests, and correlated data will indicate: this attacker wants to spam, not disclose variables or execute other code.

5. Attack mechanism

The attacker is expecting a response to this request. The response could be an HTTP 404 (not found) or 403 (forbidden) status code, indicating that formmail.pl is not available. But the response could also be an HTTP success status code such as 200 (ok), and if a vulnerable formmail.pl exists, it looks

like it might exploit it to send "w00t" in an e-mail to
sendipxx1@yahoo.com,
announcing to the attacker the spammer-friendly FormMail.

Once a spammer has found a vulnerable formmail.pl, the attacker would continue to exploit it to send spam (instead of "w00t") to targeted e-mail
addresses (instead of sendipxx1@yahoo.com).

6. Correlations:

Here are some of the apache logs in Waldo Kitty's email:

```
wks-65-30-178-154.kscable.com - - [01/Apr/2002:01:48:42 -
0500
] "GET /cgi-
bin/formmail.pl?email=f2%40aol%2Ecom&subject=www%
2Ewpusa%2Edynip%2Ecom%2Fcgi%2Dbin%2Fformmail%2Epl&recipient
=I
bTricksteri%40aol%2Ecom&msg=w00t HTTP/1.1Content-Type:
applic
ation/x-www-form-urlencoded" 403 35 "-" "Gozilla/4.0
(compati
ble; MSIE 5.5; windows 2000)"

acacdec2.ipt.aol.com - - [01/Apr/2002:04:06:22 -0500] "GET
/c
gi-
bin/formmail.pl?email=f2%40aol%2Ecom&subject=www%2Ewpusa%2
Edynip%2Ecom%2Fcgi%2Dbin%2Fformmail%2Epl&recipient=spackbiz
%4
0aol%2Ecom&msg=w00t HTTP/1.1Content-Type: application/x-
www-f
orm-urlencoded" 403 35 "-" "Gozilla/4.0 (compatible; MSIE
5.5
; windows 2000)"

clspdslgw2poold71.clsp.uswest.net - - [02/Apr/2002:09:52:12
-
0500] "GET /cgi-
bin/formmail.pl?email=f2%40aol%2Ecom&subject=
www%2Ewpusa%2Edynip%2Ecom%2Fcgi%2Dbin%2Fformmail%2Epl&recip
ie
nt=elulis%40aol%2Ecom&msg=w00t HTTP/1.1Content-Type:
applicat
ion/x-www-form-urlencoded" 403 35 "-" "Gozilla/4.0
(compatibl
e; MSIE 5.5; windows 2000)"
```

```
cm196.37.234.24.lvcm.com - - [03/Apr/2002:21:36:11 -0500]
"GE
T /cgi-
bin/formmail.pl?email=f2%40aol%2Ecom&subject=www%2Ewpu
sa%2Edynip%2Ecom%2Fcgi%2Dbin%2Fformmail%2Epl&recipient=aimj
un
kie%40yahoo%2Ecom&msg=w00t HTTP/1.1Content-Type:
application/
x-www-form-urlencoded" 403 35 "-" "Gozilla/4.0 (compatible;
M
SIE 5.5; windows 2000)"

ip68-12-222-253.ok.ok.cox.net - - [04/Apr/2002:09:20:44 -
0500
] "GET /cgi-
bin/formmail.pl?email=f2%40aol%2Ecom&subject=www%
2Ewpusa%2Edynip%2Ecom%2Fcgi%2Dbin%2Fformmail%2Epl&recipient
=s
tilispamming%40aol%2Ecom&msg=w00t HTTP/1.1Content-Type:
appli
cation/x-www-form-urlencoded" 403 35 "-" "Gozilla/4.0
(compat
ible; MSIE 5.5; windows 2000)"

msb-ts-slip22.umdnj.edu - - [05/Apr/2002:22:26:08 -0500]
"GET
 /cgi-
bin/FormMail.pl?email=lafam&subject=12%2E146%2E166%2E24
2%2Fcgi%2Dbin%2FFormMail%2Epl&recipient=thisbeastro%40aol%2
Ec
om&msg=Formmail_Found! HTTP/1.1Content-Type: application/x-
ww
w-form-urlencoded" 404 225 "-" "Gozilla/4.0 (compatible;
MSIE
 5.5; windows 2000)"

 (Kitty)
```

The same anomalies noted in our detect are evident here: missing "0d
0a"
after HTTP/1.1, "msg=w00t" and "f2@aol.com" in most entries,
and "Gozilla" User-Agent strings.
Also, the sources all seem to be hosts on dial-up,
DSL, or Cable broadband networks, just like our detect.
All these facts are also true of H D Moore's logs.
I won't copy them here, but his e-mail subject line is poignant:
"Re: SPAMMERS DELIGHT: as feeble as feeble can be." (Moore)

These bolster our assessment that this attack represents
reconnaissance
by a would-be spammer.

Michael Holstein, GCIA, found a targeted formmail.pl attack featuring actual
spam, and discussed the detect in his practical,
http://www.giac.org/practical/Michael Holstein GCIA.doc. (Holstein)

See also
http://wlofie.dyndns.org/greymatter12/archives/00000011.htm
to read a recent (October 2002) rant about these FormMail probes.
(Wlofie)

7. Evidence of active targeting:

This packet is part of a general scan,
looking for vulnerable formmail.pl scripts. The
scan is targeted at web servers, but 170.129.50.3 hasn't
yet been singled out as a vulnerable target begging the
spammer's further attention.

8 Severity:

Criticality: **(3)**
The host is a web server accessible from the public network.

Lethality: **(4)**
The protected network may suffer serious consequences, such
as blocking, blacklisting, or diminished reputation,
if it harbors a web server that spawns spam.

System Countermeasures: **(3)**
This is difficult to determine; since we do not
know any details about the web server's configuration. This median
score
is based only on unfounded speculation that /cgi-bin/formmail.pl does
not
exist on this host, or if it does, it is an up-to-date version.

Network Countermeasures: **(2)**
The firewall allows port 80 traffic to this host, so no HTTP-based exploits
will be blocked.

Criticality + Lethality - System Countermeasures - Network Countermeasures
= 2. Concern is in order, but don't panic.

intrusions@incidents.org feedback:
archived at

http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00154.html,
http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00164.html,
http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00165.html,
and
http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00172.html.

Daniel Wesserman felt
that these HTTP GETs are
so badly formed that they should give HTTP 400 errors, not 404 or 403, and that this fact makes the detect less lethal.
I had no reason to doubt Kitty's, Moore's,
or Wlofie's evidence otherwise, but I did some "telnet to port 80" experiments on some of my employer's web servers.
I discovered that while Mr. Wesserman's observation is true
of recent Apache servers, some IIS web servers don't really care how badly HTTP syntax gets mangled. I have no reason to doubt that these probes do in fact find vulnerable scripts on some web servers.

9: Defensive recommendations:

- Ensure that the web server does not
- offer a vulnerable formmail.pl.

- Review the web server access and error logs; analyze any additional
- "formmail," "Gozilla," or "w00t" substring instances found.

10: Multiple choice test question:

Consider the following web server access log entry:

```
msb-ts-slip22.umdnj.edu - - [05/Apr/2002:22:26:08 -0500]
"GET
 /cgi-
bin/FormMail.pl?email=lafam&subject=12%2E146%2E166%2E24
2%2Fcgi%2Dbin%2FFormMail%2Epl&recipient=thisbeastro%40aol%2
Ec
om&msg=Formmail_Found! HTTP/1.1Content-Type: application/x-
ww
w-form-urlencoded" 404 225 "-" "Gozilla/4.0 (compatible;
MSIE
 5.5; windows 2000)"
```

With which of the following Snort alert messages would this entry most likely
correlate?

a) WEB-CGI formmail access

b) WEB-CGI phf access

c) WEB-ATTACKS /usr/bin/id command attempt

c) WEB-ATTACKS mail command attempt

(answer: a)

References

DigitalCandle. "Go!Zilla." Home page. 2002.
URL:http://gozilla.com/ (29 Jan. 2003)

Holstein, Michael. "SANS GCIA Practical Assignment." SANS GIAC.
URL:http://www.giac.org/practical/Michael_Holstein_GCIA.doc (29 Jan.
2003)

IETF. "Hyperttext Transfer Protocol -- HTTP/1.1." RFC 2068. Jan. 1997.
URL:http://www.ietf.org/rfc/rfc2068.txt (29 Jan. 2003)

Kitty, Waldo. "Re: Is open proxies something new (in SpamCop)?"
E-mail to SpamCop-List. 10 Apr. 2002.
URL:http://news.spamcop.net/pipermail/spamcop-list/2002-
April/000254.html
(29 Jan. 2003)

Latis. "StillSecure." Latis Networks home page. 2002.
URL:http://www.latis.com/ (29 Jan. 2003)

Moore, H D. "Re: SPAMMERS DELIGHT: as feeble as feeble can be." E-mail
to "Mailing List Mobile Code." 11 Dec 2001.
URL:http://citadelle.intrinsec.com/mailing/current/HTML/ml_mobile_code/
0487.html (29 Jan. 2003)

Snort. "Snort, The Open Source Network Intrusion Detection System."
URL:http://www.snort.org/ (29 Jan. 2003)

Wlofie. "formmail is in season." wlofie'z GrayMatter blog. 1 Oct. 2002.
URL:http://wlofie.dyndns.org/greymatter12/archives/00000011.htm
(29 Jan. 2003)

Wright, Matt. "Matt's Script Archive: FormMail." 21 Apr. 2002.
URL:http://www.scriptarchive.com/formmail.html (29 Jan. 2003)

Part 3

``Analyze This''

To: Director of Network Security, Practical University

From: Carl Gibbons

Subject: Analysis of 30 Dec 2002 - 3 Jan 2003 Snort Logs

Executive Summary

At your instruction, I obtained the files listed below,
and confirmed that they contain Snort data collected at PU
between
Monday 30 December 2002 and Friday 3 January 2003 inclusive.

file name	size (bytes)	timestamp
OOS_Report_2002_12_31_10852	665,603	Tue Dec 31 00:05:23 2002
alert.021230.gz	593,661	Fri Jan 3 05:00:44 2003
scans.021230.gz	2,695,953	Fri Jan 3 05:00:46 2003
OOS_Report_2003_01_01_19650	337,923	Wed Jan 1 00:05:17 2003
alert.021231.gz	2,518,917	Sat Jan 4 05:00:47 2003
scans.021231.gz	13,579,786	Sat Jan 4 05:01:04 2003
OOS_Report_2003_01_02_2030	240,643	Thu Jan 2 00:05:23 2003
alert.030101.gz	2,447,686	Sun Jan 5 05:00:49 2003
scans.030101.gz	9,849,273	Sun Jan 5 05:01:01 2003
OOS_Report_2003_01_03_17638	460,803	Fri Jan 3 00:05:21 2003
alert.030102.gz	1,097,007	Mon Jan 6 05:04:03 2003
scans.030102.gz	5,079,538	Mon Jan 6 05:04:38 2003
OOS_Report_2003_01_04_29001	312,323	Sat Jan 4 00:05:19 2003
alert.030103.gz	1,380,119	Tue Jan 7 05:00:54 2003
scans.030103.gz	6,035,144	Tue Jan 7 05:01:04 2003

I have analyzed these
and organized the attached report as follows.

- Statistcal Overview
-
  - Data volume and distribution
  - Non-targeted scanning activity by remote hosts
  - Targeted scanning activity by remote hosts
  - Alert summary
-
- Top ten events of interest ("top talkers")
-
  1. Alerts: Russia Dynamo, 194.87.6.75 <--> MY.NET.105.204
  2. Alerts: IIS Unicode attacks, MY.NET.112.204 ---> 61.236.39.3
  3. Alerts: IIS Unicode attacks, 209.196.6.226 ---> 30 PU targets
  4. Alerts: IIS Unicode attacks, 5 remote hosts ---> MY.NET.70.207
  5. Alerts: IIS Unicode and Nimda attacks,
  6. MY.NET.71.230 ---> 1,569 remote targets
  7. Alerts: SMB Name Wildcard, 81.50.52.235 ---> 862 PU hosts,
  8. and SMB Name Wildcard and SMB C Access,
  9. many remote and local hosts
  10. Alerts: High port 65535 tcp, 72 remote hosts <--> 17 PU hosts
  11. Alerts: TFTP UDP connections, 5 PU TFTP servers ---> 192.168.0.253
  12. Alerts: Watchlist 000220 IL-ISDNNET-990517,
  13. many remote and local hosts
  14. Scans/OOS: peer-to-peer (P2P) file sharing,
  15. many remote and local hosts
-

- Tools and processes used in this analysis
-
    - snort-rep and "Perl Munging."
    - Post-analysis

Defensive recommendations are included throughout the report
for each item outlined,
and for your convenience are marked with boldface
Defensive recommendation prefixes.
Your security team should study and verify this report,
and compare its findings with your own
and those of your other consultants.
Thank you! - CG

Statistical Overview

Data volume and distribution

Table 1: log entries

842,113	entries in alert logs
43	distinct alert types in alert logs (excluding portscan alerts)
4,599,575	entries in scan logs
6,254	entries in "Out-Of-Spec" (OOS) logs

Of the alert file entries, 841,985 alerts were processed using
automated tools, but 148 malformed entries had to be inspected
manually.
The malformed entries suggest that
there may be a problem with your Snort alert logging architecture,
or possibly a problem in the Snort code itself.
See the "Tools and Processes" section for details.

Monday's scan and alert log files have an eighteen hour gap
of no logged data between 00:40 and 18:38. I shall assume
that your host that records Snort alert logs was offline,
possibly for end-of-year maintenance, during this period.

The volume of log data is consistent with what
I expect of holiday time, "between semesters" traffic at
a University. The largest alert and scan logs are those of
New Year's Eve and New Year's Day.

For each alert and scan file,
there is a delay (approximately three days five hours)
between the timestamp of the last event in that file
and the timestamp of the file itself. This may be
the result of an improperly synchronized system clock
on a machine that handled these files, but
could also be the result of regular or automated tampering
of these files.
Therefore, unless chain-of-custody issues regarding these data are
resolved,
I don't recommend using the information in this report as evidence in
support
of disciplinary or legal action against any individual or entity.
Defensive recommendation:
review procedures for handling log data.

Non-targeted scanning activity by remote hosts

Table 2: portscan breakdown

20 remote hosts scanned 3000 or more PU hosts or ports,

```
 52 remote hosts scanned 300-2999 PU hosts or ports,
203 remote hosts scanned 30-299 ports,
212 remote hosts scanned 30-299 PU hosts,
233 remote hosts scanned 29 or fewer ports,
226 remote hosts scanned 29 or fewer PU hosts,
508 portscanning remote hosts detected in total.
```

Table 3: top twenty portscanning remote hosts:

ports	hosts	remote host
72370	68285	80.14.115.177
20850	20207	150.187.177.12
18927	13049	81.50.52.235
11517	11313	202.181.214.4
9802	9562	194.248.237.100
8778	8474	67.113.252.218
8475	8377	208.176.192.14
8030	4	24.201.216.239
7765	6786	80.200.151.134
7508	7281	147.134.96.17
6580	6457	208.8.19.34
6010	5937	193.194.50.9
5923	5800	218.145.194.187
5864	5804	62.4.16.140
5847	5762	204.215.188.182
5711	5678	12.164.194.254
4959	4607	24.201.118.234
4847	4587	194.15.164.33
4519	4369	210.255.49.131
3957	3878	209.196.6.226

All of the top twenty were systematic scans of PU hosts, looking for the following potentially exploitable services:
Microsoft Networking/NBT/CIFS (TCP ports 135, 139, 445; UDP port 137),
Radmin (TCP port 4899),
web servers (TCP ports 80, 443),

Microsoft Terminal Services (TCP port 3389),
ssh (port 22),
SOCKS (port 1080),
and Microsoft SQL Server (TCP port 1433).
Defensive recommendation:
a properly configured firewall and regular vulnerability assessment scanning
are desirable countermeasures against such reconnaissance.

Most of the top twenty portscanners were also taking advantage of the holiday; their probes occurred between 07:42 Tuesday and 13:08:55 Thursday. Perhaps they are hoping that their holiday scans will pass unnoticed.

Targeted scanning activity by remote hosts

Of particular concern are the remote hosts that targeted particular PU hosts for extensive port scanning. Let's look at five of these incidents in detail.

Table 4: portscan events by remote hosts
probing an average of 10 or more ports per scanned PU host:

ports	hosts	remote host	ratio
8030	4	24.201.216.239	2007.5
116	1	208.185.54.31	116
29	1	194.176.61.59	29
122	5	131.118.254.1	24
65	5	193.163.220.4	13

The scans.030103.gz logs show
that the remote machine 24.201.216.239
(modemcable239.216-201-24.que.mc.videotron.ca)
probed TCP ports 1-4992
of MY.NET.150.213 between 21:17 and 21:23 on Friday:

```
Jan  3 21:17:44 24.201.216.239:1096 -> MY.NET.150.213:1 SYN ******S*
Jan  3 21:17:44 24.201.216.239:1099 -> MY.NET.150.213:4 SYN ******S*
```

```
Jan  3 21:17:43 24.201.216.239:1102 -> MY.NET.150.213:7 SYN ******S*
Jan  3 21:17:44 24.201.216.239:1103 -> MY.NET.150.213:8 SYN ******S*
...
Jan  3 21:23:20 24.201.216.239:2572 -> MY.NET.150.213:4991 SYN ******S*
Jan  3 21:23:20 24.201.216.239:2573 -> MY.NET.150.213:4992 SYN ******S*
Jan  3 21:23:20 24.201.216.239:2565 -> MY.NET.150.213:4985 SYN ******S*
```

The alert.0301030.gz logs also show that a TFTP server on
MY.NET.150.213 responded affirmatively to these TCP probes,
so this event and the involved PU host merit attention:

```
01/03-21:17:47.261629  [**]
TFTP - External TCP connection to internal tftp server [**]
24.201.216.239:1164 -> MY.NET.150.213:69

01/03-21:17:47.261815  [**]
TFTP - External TCP connection to internal tftp server [**]
MY.NET.150.213:69 -> 24.201.216.239:1164
```

Other alerts show that another remote host,
12.240.219.52, also elicited TFTP responses from UDP probes
to MY.NET.150.213 at 05:34 that same day.
There are viruses/worms that run TFTP servers
to propagate themselves or provide backdoors on infected hosts;
MY.NET.150.213 may be infected.

Also, the alerts show that many times on Thursday and Friday
it exchanged UDP port 65535 datagrams with seventeen different outside
hosts,
but these exchanges may be WinMX (http://www.winmx.com)
peer-to-peer file sharing traffic, as they are all between
WinMX's local port 6257 and remote port 65535.

Defensive recommendation:

please consider MY.NET.150.213 compromised, and
quarantine it for further investigation by your security team.

The remote host 208.185.54.31
(208.185.54.31.speedera.com)
probed random UDP ports at MY.NET.112.210.
There is no evidence in the logs that the target responded.

The remote host 194.176.61.59 probed MY.NET.53.31
for many well known trojan TCP ports,
such as 31337 (BackOrifice) and 12345 (Netbus).
These were definitely hostile scans,
but there is no evidence in the logs that the target responded.

Defensive recommendation:

Please report these hostile scans to the Lithuanian address reported by

RIPE Whois (RIPE):

```
inetnum:      194.176.61.0 - 194.176.61.255
netname:      OMNITEL-CUSTOMERS
descr:        OMNITEL-CUSTOMERS
country:      LT
admin-c:      ODA1-RIPE
tech-c:       ODA1-RIPE
rev-srv:      ns1.omnitel.net
rev-srv:      ns2.omnitel.net
status:       ASSIGNED PA
notify:       hostmaster@omnitel.net
mnt-by:       AS5522-MNT
changed:      m.malakauskas@omnitel.net 20020613
source:       RIPE

route:        194.176.61.0/24
descr:        OMNITEL customer block
origin:       AS5522
notify:       hostmaster@omnitel.net
mnt-by:       AS5522-MNT
changed:      m.malakauskas@omnitel.net 20020717
source:       RIPE

person:       OMNITEL DNS Administrator
address:      T.Sevcenkos 25
address:      Vilnius 2600
address:      Lithuania
phone:        +370 2 221712
fax-no:       +370 2 220627
e-mail:       hostmaster@omnitel.net
nic-hdl:      ODA1-RIPE
notify:       hostmaster@omnitel.net
changed:      s.kazlauskas@omnitel.net 19981207
source:       RIPE
```

The DNS server 131.118.254.1
(ns.ums.edu)
had a habit of rapidly spraying UDP datagrams
from source port 53 (DNS) at several consecutive ports of
MY.NET.137.46. This is not customary DNS server behavior.
Perhaps these are responses to stimuli spoofed with MY.NET.137.46's
address.
Throughout the week there are several
Microsoft Networking (SMB Name Wildcard, SMC C access)
alerts on traffic to MY.NET.137.46, but nothing that correlates with
the port 53 bursts.

The scanner at 193.163.220.4
(proxy-scanner.eris.dk)
rapidly probed each of
 MY.NET.83.188, MY.NET.60.40, and MY.NET.27.210 for ports

80, 8080, 8081, 3128, 1080, 6552, 5104, 5262, 4438, 5634, 7113, 7464,
and 23,
in that order.
This scanning pattern is common among IRC server operators.
There are 131 IRC entries in the alert logs
that correlate with the probes to MY.NET.83.188.

Alert summary

Table 5: Alert instance counts (excluding portscans)

%	#	description
42.8	156445	Russia Dynamo - SANS Flash 28-jul-00
13.8	50323	spp_http_decode: IIS Unicode attack detected
13.5	49282	SMB Name Wildcard
13.4	48854	High port 65535 tcp - possible Red Worm - traffic
7.4	27022	TFTP - External UDP connection to internal tftp server
2.9	10530	Watchlist 000220 IL-ISDNNET-990517
2.5	9106	NIMDA - Attempt to execute cmd from campus host
0.7	2647	High port 65535 udp - possible Red Worm - traffic
0.6	2033	spp_http_decode: CGI Null Byte attack detected
0.5	1943	Queso fingerprint
0.4	1354	Watchlist 000222 NET-NCFC
0.4	1336	NIMDA - Attempt to execute root from campus host
0.3	1027	IRC evil - running XDCC
0.2	849	External RPC call
0.2	587	SUNRPC highport access!
0.1	385	Tiny Fragments - Possible Hostile Activity
0.1	283	Incomplete Packet Fragments Discarded
0.1	275	Null scan!
0.1	184	SMB C access
0.0	157	Port 55850 tcp - Possible myserver activity - ref. 010313-1
0.0	144	TFTP - Internal UDP connection to external tftp server
0.0	137	EXPLOIT x86 NOOP
0.0	84	Possible trojan server activity
0.0	81	NMAP TCP ping!
0.0	56	EXPLOIT x86 setuid 0
0.0	35	EXPLOIT x86 setgid 0
0.0	26	Attempted Sun RPC high port access
0.0	18	TCP SRC and DST outside network
0.0	16	Port 55850 udp - Possible myserver activity - ref. 010313-1
0.0	15	FTP passwd attempt
0.0	12	RFB - Possible WinVNC - 010708-1
0.0	10	TFTP - External TCP connection to internal tftp server
0.0	9	EXPLOIT x86 stealth noop

```
 0.0      3      EXPLOIT NTPDX buffer overflow
 0.0      2      ICMP SRC and DST outside network
 0.0      2      IDS552/web-iis_IIS ISAPI Overflow ida nosize
 0.0      2      Probable NMAP fingerprint attempt
 0.0      1      Back Orifice
 0.0      1      External FTP to HelpDesk MY.NET.70.49
 0.0      1      External FTP to HelpDesk MY.NET.70.50
 0.0      1      SYN-FIN scan!
 0.0      1      connect to 515 from outside

==========================================================================
```

Even though each of these 43 alerts represent potential network abuse,
not all of them are discussed here.
With one exception (peer-to-peer traffic),
each of the top ten events of interest
correspond to one of the top seven alerts.
However, the analyses below
also include discussion of many of the less common alerts,
when they occur concurrently with a "top-ten talker's" attack,
or help to explain one.

The following three tables identify the
top alert generators among remote hosts, local hosts, and local
port numbers. These were used to help identify the top
ten events of interest ("top talkers").

Table 6: Alert summary, remote hosts
detected in more than 1000 instances of a triggered alert:

```
==========================================================================
     # remote host          description
==========================================================================
156445 194.87.6.75          Russia Dynamo - SANS Flash 28-jul-00
 27021 192.168.0.253        TFTP - External UDP connection to internal tftp
                            server
 26797 61.236.39.3          spp_http_decode: IIS Unicode attack detected
 14818 217.136.65.154       High port 65535 tcp - possible Red Worm - traffic
  8452 81.50.52.235         SMB Name Wildcard
  7882 80.200.151.134       High port 65535 tcp - possible Red Worm - traffic
  5285 217.136.157.114      High port 65535 tcp - possible Red Worm - traffic
```

```
  5034 80.200.117.120       High port 65535 tcp - possible Red Worm - traffic
  3196 172.179.155.153      High port 65535 tcp - possible Red Worm - traffic
  2890 209.196.6.226        spp_http_decode: IIS Unicode attack detected
  2848 80.15.81.185         High port 65535 tcp - possible Red Worm - traffic
  2161 212.179.96.28        Watchlist 000220 IL-ISDNNET-990517
  1818 62.23.74.165         spp_http_decode: IIS Unicode attack detected
  1757 212.179.104.42       Watchlist 000220 IL-ISDNNET-990517
  1522 212.179.103.3        Watchlist 000220 IL-ISDNNET-990517
  1396 212.179.107.228      Watchlist 000220 IL-ISDNNET-990517
  1044 217.128.204.221      High port 65535 tcp - possible Red Worm - traffic
```

Table 7: Alert summary, local hosts
detected in more than 1000 instances of a triggered alert:

```
================================================================================
     # local host          description
================================================================================
156445 MY.NET.105.204      Russia Dynamo - SANS Flash 28-jul-00
 26798 MY.NET.112.204      spp_http_decode: IIS Unicode attack detected
 25957 MY.NET.84.151       High port 65535 tcp - possible Red Worm - traffic
 19919 MY.NET.88.193       High port 65535 tcp - possible Red Worm - traffic
  9104 MY.NET.71.230       NIMDA - Attempt to execute cmd from campus host
  5418 MY.NET.111.235      TFTP - External UDP connection to internal tftp
                           server
  5414 MY.NET.111.232      TFTP - External UDP connection to internal tftp
                           server
  5410 MY.NET.111.219      TFTP - External UDP connection to internal tftp
                           server
  5393 MY.NET.111.231      TFTP - External UDP connection to internal tftp
                           server
  5386 MY.NET.111.230      TFTP - External UDP connection to internal tftp
                           server
  4574 MY.NET.71.230       spp_http_decode: IIS Unicode attack detected
  2907 MY.NET.91.104       Watchlist 000220 IL-ISDNNET-990517
  2580 MY.NET.198.220      High port 65535 tcp - possible Red Worm - traffic
  2073 MY.NET.70.207       spp_http_decode: IIS Unicode attack detected
  1957 MY.NET.84.193       Watchlist 000220 IL-ISDNNET-990517
  1806 MY.NET.113.4        Watchlist 000220 IL-ISDNNET-990517
  1724 MY.NET.183.26       spp_http_decode: IIS Unicode attack detected
  1486 MY.NET.84.133       spp_http_decode: IIS Unicode attack detected
  1336 MY.NET.71.230       NIMDA - Attempt to execute root from campus host
  1066 MY.NET.70.176       High port 65535 udp - possible Red Worm - traffic
  1050 MY.NET.83.146       High port 65535 udp - possible Red Worm - traffic
```

Table 8:
Alert summary, local ports associated with 100 or more alerts

#	port	description
165	ftp	Watchlist 000222 NET-NCFC
802	smtp	Queso fingerprint
8257	http	spp_http_decode: IIS Unicode attack detected
1071	http	Queso fingerprint
419	http	Watchlist 000222 NET-NCFC
849	sunrpc	External RPC call
49282	netbios-ns	SMB Name Wildcard
184	netbios-ssn	SMB C access
22530	1191	Russia Dynamo - SANS Flash 28-jul-00
4146	1214	Watchlist 000220 IL-ISDNNET-990517
307	1237	Watchlist 000220 IL-ISDNNET-990517
1957	2418	Watchlist 000220 IL-ISDNNET-990517
844	2435	Watchlist 000220 IL-ISDNNET-990517
609	2502	Watchlist 000220 IL-ISDNNET-990517
191	2515	Watchlist 000222 NET-NCFC
552	2681	Watchlist 000220 IL-ISDNNET-990517
10117	2718	Russia Dynamo - SANS Flash 28-jul-00
1561	3144	Russia Dynamo - SANS Flash 28-jul-00
23977	3514	Russia Dynamo - SANS Flash 28-jul-00
569	4125	Watchlist 000220 IL-ISDNNET-990517
7100	4930	Russia Dynamo - SANS Flash 28-jul-00
1177	6257	High port 65535 udp - possible Red Worm - traffic
587	32771	SUNRPC highport access!
26813	65535	High port 65535 tcp - possible Red Worm - traffic

Top ten events of interest ("top talkers")

Alerts: Russia Dynamo, 194.87.6.75 <--> MY.NET.105.204

In July 2000, SANS reported a widespread infestation of hostile code which sent traffic to the 194.87.6.0/24 (*.dynamic.dol.ru) network.
See

http://archives.neohapsis.com/archives/sans/2000/0068.html (SANS[1])
and
http://www.sans.org/y2k/072818.htm (Northcutt)
for background information on these alerts.
Miika Turkia, GCIA (Turkia), who also studied a previous year's
end-of-year alerts,
also noticed this traffic and called it "weird." He
suggested that the local host may have been Trojan-horse compromised.

Traffic between MY.NET.105.204 and 194.87.6.75
(75.6.87.194.dynamic.dol.ru)
occurs in constant streams for the following intervals:

- 20:57-23:17 New Year's Eve (local port 3514)
- 03:53-05:42 New Year's Day (local port 1191)
- 05:53-06:26 New Year's Day (local port 4930)
- 07:46-08:31 New Year's Day (local port 2718)
- 08:35-08:42 New Year's Day (local port 3144)

If you are paranoid, examine MY.NET.105.204 for signs of a Trojan.
But it's more likely that these alerts represent
five harmless sessions of streaming traffic,
such as a video conference, between a Russian student stuck
at PU for the holiday, and the dynamic.dol.ru customer for
whom he or she is homesick.

Alerts: IIS Unicode attacks, MY.NET.112.204 ---> 61.236.39.3

Bugtraq id 1806
(SecurityFocus Online)
discusses these attacks in detail.
Attackers send carefully written URLs to an unpatched Microsoft IIS
hosts,
and the results can include unauthorized disclosure, modification,
deletion,
or execution of files.

The Bugtraq alert warns of worms
loose in the wild that explioit this vulnerability.
These are alerts we should take seriously.

Microsoft has revised their security bulletin
MS00-078 (Microsoft Technet)
regarding this alert. They have released a patch that not only
fixes this vulnerability, but another as well.
Defensive recommendation:
Please ensure that all PU-owned servers that run IIS are patched.

There were 1811 distinct remote hosts and
428 distinct local hosts among the 50,323 "IIS Unicode" alerts.
More than half (26,797; 53%) were caused by relentless
traffic from MY.NET.112.204 to 61.236.39.3,
starting at 11:46 on New Year's Eve and continuing
(seldom interrupted) through 00:07 Friday morning.
Return traffic from 61.236.39.3 also
triggered four "EXPLOIT x86 set[ug]id" alerts at
widely spaced intervals during the same time period,
which increases my suspicion that these alerts represent
malicious traffic.
Defensive recommendation:
Please quarantine and investigate MY.NET.112.204.
If you wish to contact the remote network administrator about this
event,
APNIC Whois
has the following records regarding 61.236.39.3:

```
inetnum:      61.232.0.0 - 61.237.255.255
netname:      CRTC
country:      CN
descr:        CHINA RAILWAY TELECOMMUNICATIONS CENTER
admin-c:      LQ112-AP
tech-c:       LM273-AP
status:       ALLOCATED PORTABLE
changed:      ipas@cnnic.net.cn 20030121
mnt-by:       MAINT-CNNIC-AP
source:       APNIC

person:       LV QIANG
nic-hdl:      LQ112-AP
e-mail:       lq@crc.net.cn
address:      22F Yuetan Mansion,Xicheng District,Beijing,P.R.China
phone:        +86-10-51890499
fax-no:       +86-10-51890674
country:      CN
```

```
changed:      ipas@cnnic.net.cn 20030121
mnt-by:       MAINT-CNNIC-AP
source:       APNIC

person:       liu min
nic-hdl:      LM273-AP
e-mail:       mliu@crc.net.cn
address:      22F Yuetan Mansion,Xicheng District,Beijing,P.R.China
phone:        +86-10-51848796
fax-no:       +86-10-51842426
country:      CN
changed:      ipas@cnnic.net.cn 20030121
mnt-by:       MAINT-CNNIC-AP
source:       APNIC

mntner:       MAINT-CNNIC-AP
descr:        Computer Network Information Center
descr:        Chinese Academy of Science
admin-c:      HQ1-CN
tech-c:       MW1-AP
upd-to:       dbmon@apnic.net
auth:         NONE
notify:       dbmon@apnic.net
mnt-by:       MAINT-CNNIC-AP
referral-by:  MAINT-NULL
changed:      hostmaster@apnic.net 970408
source:       APNIC

person:       Hualin Qian
address:      Chinese Academy of Sciences
address:      Computer Network Center
address:      P.O.Box 2418-26
address:      Beijing, 100081
address:      CN
phone:        +86 1 2569960
e-mail:       hlqian@ns.cnc.ac.cn
nic-hdl:      HQ1-CN
notify:       dbmon@apnic.net
mnt-by:       MAINT-NULL
changed:      hostmaster@apnic.net 19950419
source:       APNIC

person:       Mao Wei
address:      China Internet Information Center(CNNIC)
  No. 4 of South street ,Zhongguancun,Haidian District
address:      Beijing,100080
address:      P.R.China
country:      CN
phone:        +86-10-62619750
fax-no:       +86-10-62559892
e-mail:       mao@cnnic.net.cn
nic-hdl:      MW1-AP
mnt-by:       MAINT-CNNIC-AP
changed:      IPAS@CNNIC.NET.CN 20010319
source:       APNIC
```

Alerts: IIS Unicode attacks, 209.196.6.226 ---> 30 PU targets

Remote host 209.196.6.226 (226-209.196.6.dellhost.com)
triggered this alert 2,890
times, and that host is also a top twenty remote portscanner.
ARIN Whois identifies
Atlanta-based Interliant as responsible for 209.196.6.0/24:

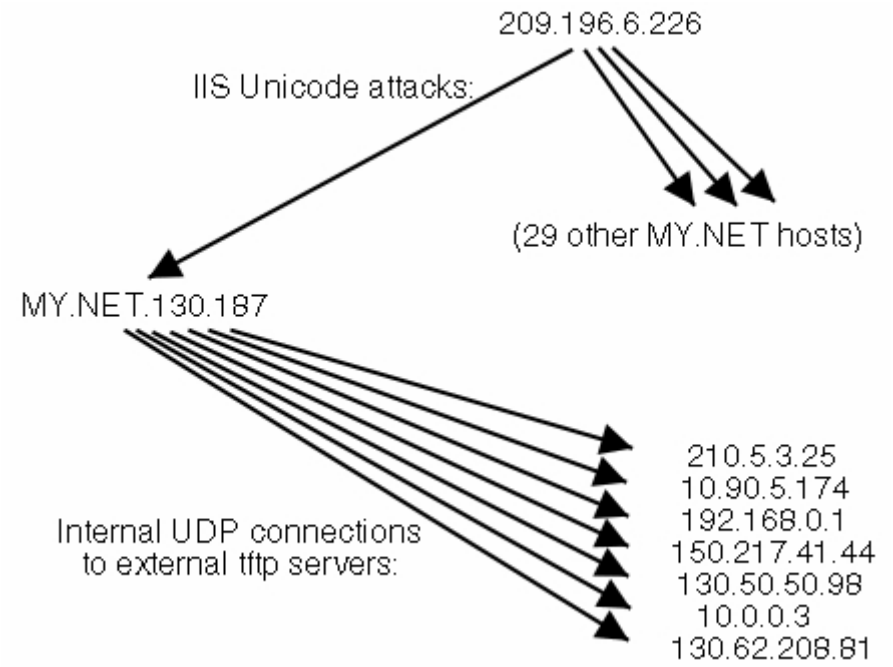
```
OrgName:    Interliant
OrgID:      INTERL-38

NetRange:   209.196.6.0 - 209.196.6.255
CIDR:       209.196.6.0/24
NetName:    ILNT-DH37
NetHandle:  NET-209-196-6-0-1
Parent:     NET-209-196-0-0-1
NetType:    Reassigned
NameServer: NS1.US.DELLHOST.COM
Comment:
RegDate:    2001-08-17
Updated:    2001-08-17

TechHandle: AG138-ARIN
TechName:   Galiano, Aj
TechPhone:  +1-770-673-2202
TechEmail:  neteng@sagenetworks.com
```

This remote machine alerted
on traffic to each of the following 30 PU hosts no fewer than 20 times,
so the attacks may have been focused on these specific PU targets.
Defensive recommendation:
Contact Interliant and ask about these scans.

MY.NET.5.92	MY.NET.91.8	MY.NET.113.208	MY.NET.130.122
MY.NET.5.95	MY.NET.91.154	MY.NET.130.14	MY.NET.130.187
MY.NET.11.2	MY.NET.104.177	MY.NET.130.27	MY.NET.132.42
MY.NET.29.3	MY.NET.105.204	MY.NET.130.34	MY.NET.136.2
MY.NET.29.10	MY.NET.106.191	MY.NET.130.40	MY.NET.150.101
MY.NET.70.207	MY.NET.106.222	MY.NET.130.86	MY.NET.157.12
MY.NET.83.248	MY.NET.110.76	MY.NET.130.91	MY.NET.157.52
MY.NET.86.19	MY.NET.111.21		

Of these 30 hosts, MY.NET.130.187 triggers "Internal UDP
connection to external tftp server" for seven distinct remote
addresses,
three of which are RFC-1918 addresses (see illustration).
This is behavior consistent with
an infected machine trying to propagate its virus via TFTP.
Defensive recommendation:
Please quarantine this host and investigate whether it is compromised.

Code Blue (Sophos[1])
is one example of code that exploits the IIS Unicoce vulnerability
and propagates via TFTP. I don't believe MY.NET.130.187 is
infected with Code Blue, because I don't see other
signatures of Code Blue infection, such as attacks
by infected hosts at 211.167.67.167 (www.nsfocus.com).
But it may be infected with a variant.



Also among the other 29 targets,
MY.NET.70.207 traffic represents another event of interest,
discussed next.

Alerts: IIS Unicode attacks, 5 remote attackers ---> MY.NET.70.207

Not only did the dellhost.com machine at 209.196.6.226
attack MY.NET.70.207 (see above),
but four others did as well.

62.23.74.165 (host.165.74.23.62.rev.coltfrance.com)
triggered this alert 1,818 times
in a 5½ minute interval beginning at 05:50 New Year's day.
Also, 62.226.25.106 (p3EE2196A.dip.t-dialin.net),

195.71.226.156 (tomp-gtso-de01.mediaways.net),
and 65.203.113.240 all sent rapid
bursts to MY.NET.70.207's port 80 at different times.
Defensive recommendation:
If this is an IIS server, please check its web access logs for activity
by these attackers.

Alerts: IIS Unicode and Nimda attacks,

MY.NET.71.230 ---> 1,569 remote targets

Not only did MY.NET.71.230 trigger 4,574 "IIS Unicode" alerts,
but it also triggered 10,440 Nimda alerts.

See the
Sophos virus analysis of Nimda-D (Sophos[2]) for more information.
MY.NET.71.230 is apparently scanning remote hosts, looking for
Nimda or Code Red II (Sophos[3]) compromised hosts that allow remote
execution of
commands via cmd.exe or root.exe.
(Nimda variants also try to spread themselves this way.)
Defensive recommendation:
MY.NET.71.230 is almost certainly compromised;
please quarantine and clean this machine.

Alerts: SMB Name Wildcard, 81.50.52.235 ---> 862 PU hosts,

and SMB Name Wildcard and SMB C Access,

many remote and local hosts

These triggered for 860 distinct remote hosts and 925 distinct local
hosts.
SMB Wildcard queries are ubiquitous in traffic among Microsoft windows
hosts.
Defensive recommendation:
No NBT or CIFS traffic (ports 135-139, 445) should be allowed
through the campus firewall, because the protocol itself exposes
machines
to many security and privacy vulnerabilities.

The top triggerer is 81.50.52.235 (ALille-208-1-17-235.abo.wanadoo.fr),
which is also one of the top twenty portscanners.

It scanned for port 137 (Microsoft
NetBIOS) at 862 distinct addresses in PU's MY.NET.132/22 and
MY.NET.137/24
networks, so while it wasn't targeted at particular hosts, it does
appear
to be a targeted scan at particular subnets. This host also tried
15 FTP passwd attempt attacks on New Year's Eve at 8 distinct PU hosts.

RIPE Whois (RIPE)
reports the following registration for this attacker's network.
Defensive recommendation: report these attacks to
abuse@wanadoo.fr.

```
inetnum:      81.50.52.0 - 81.50.52.255
netname:      IP2000-ADSL-BAS
descr:        BSLIL208 Lille Bloc1
country:      FR
admin-c:      WITR1-RIPE
tech-c:       WITR1-RIPE
status:       ASSIGNED PA
remarks:      for hacking, spamming or security problems send mail to
remarks:      postmaster@wanadoo.fr AND abuse@wanadoo.fr
remarks:      for ANY problem send mail to
gestionip.ft@francetelecom.com
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20021120
source:       RIPE

route:        81.50.0.0/16
descr:        France Telecom
descr:        Wanadoo Interactive
remarks:      ----------------------------------------
remarks:      For Hacking, Spamming or Security problems
remarks:      send mail to   abuse@wanadoo.fr
remarks:      ----------------------------------------
origin:       AS3215
mnt-by:       RAIN-TRANSPAC
mnt-routes:   RAIN-TRANSPAC
changed:      tom@rain.fr 20021030
source:       RIPE

role:         Wanadoo Interactive Technical Role
address:      WANADOO INTERACTIVE
address:      48 rue Camille Desmoulins
address:      92791 ISSY LES MOULINEAUX CEDEX 9
address:      FR
phone:        +33 1 58 88 50 00
e-mail:       abuse@wanadoo.fr
e-mail:       postmaster@wanadoo.fr
admin-c:      FTI-RIPE
tech-c:       TEFS1-RIPE
nic-hdl:      WITR1-RIPE
notify:       gestionip.ft@francetelecom.com
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20010504
```

```
changed:      gestionip.ft@francetelecom.com 20010912
changed:      gestionip.ft@francetelecom.com 20011204
source:       RIPE
```

The SMB C Access alerts are troubling.
This alert occurred 184 times, on 90 remote and 15 local hosts.
Defensive recommendation:
You should be paranoid about any
access of "shares" (shared folders, drives, printers, etc.)
through the firewall.
Consider deploying a VPN solution for the small number of users
that absolutely must have this functionality.

Alerts: High port 65535 tcp, 72 remote hosts <--> 17 PU hosts

Adore (Sophos[4]), also known as
Red Worm
(SANS[2]),
exploits several Linux vulnerabilities.
One of the side effects of a compromised Linux box is a backdoor
root shell listening on port 65535, which may be activated if
the packets it receives are a certain size.

The alerts don't show the sizes of the tcp segments that triggered
them.
Without more information,
it is difficult to guess whether your TCP port 65535 traffic is
malicious or merely unusual. You could take the drastic
step of blocking all port 65535 traffic, but that
move may bring other undesirable side effects.

TCP port 65535 does not appear in any scans.*.gz entry.
Nevertheless, one of the top twenty portscanners,
80.200.151.134
(134.151-200-80.adsl.skynet.be), also triggered 7,882 "High port"
alerts.
It did so between Thursday 07:35 and Friday 15:53, in traffic to
TCP port 65535 of three distinct PU hosts:
MY.NET.84.151, MY.NET.88.193, and MY.NET.198.220.
Defensive recommendation:

Report this activity as possible but unconfirmed network abuse.
RIPE Whois (RIPE) has Skynet Belgium's contact information:

```
inetnum:      80.200.0.0 - 80.200.255.255
netname:      BE-SKYNET-20011108
descr:        ADSL Customers
descr:        Skynet Belgium
country:      BE
admin-c:      JFS1-RIPE
tech-c:       PDH16-RIPE
status:       ASSIGNED PA
mnt-by:       SKYNETBE-MNT
changed:      ripe@skynet.be 20011212
source:       RIPE

route:        80.200.0.0/15
descr:        SKYNETBE-CUSTOMERS
origin:       AS5432
notify:       noc@skynet.be
mnt-by:       SKYNETBE-MNT
changed:      noc@skynet.be 20011116
source:       RIPE

person:       Jean-Francois Stenuit
address:      Belgacom Skynet NV/SA
address:      Rue Carli 2
address:      B-1140 Bruxelles
address:      Belgium
phone:        +32 2 706-1311
fax-no:       +32 2 706-1150
e-mail:       jfs@skynet.be
nic-hdl:      JFS1-RIPE
remarks:      ----------------------------------------
remarks:      Network problems to: noc@skynet.be
remarks:      Peering requests to: peering@skynet.be
remarks:      Abuse notifications to: abuse@skynet.be
remarks:      ----------------------------------------
mnt-by:       SKYNETBE-MNT
changed:      jfs@skynet.be 19970707
changed:      ripe@skynet.be 20021125
source:       RIPE

person:       Pieterjan d'Hertog
address:      Belgacom Skynet sa/nv
address:      2 Rue Carli
address:      B-1140 Brussels
address:      Belgium
phone:        +32 2 706 13 11
fax-no:       +32 2 706 13 12
e-mail:       piet@skynet.be
nic-hdl:      PDH16-RIPE
remarks:      ----------------------------------------
remarks:      Network problems to: noc@skynet.be
remarks:      Peering requests to: peering@skynet.be
remarks:      Abuse notifications to: abuse@skynet.be
remarks:      ----------------------------------------
mnt-by:       SKYNETBE-MNT
```

```
changed:      jfs@skynet.be 19990415
changed:      piet@skynet.be 19991210
changed:      piet@skynet.be 20000302
changed:      piet@skynet.be 20020329
source:       RIPE
```

Alerts: TFTP UDP connections, 5 PU TFTP servers ---> 192.168.0.253

These TFTP connections happen at regular intervals during the 5 day period.
All five PU servers are in the same /24 subnet; they are
MY.NET.111.219,
MY.NET.111.230,
MY.NET.111.231,
MY.NET.111.232, and
MY.NET.111.235.

The recipient of these datagrams has an RFC 1918 private address; if that host has no other network interfaces or configurations, then it is not a remote host on the public internet. Logically, it is a host on PU's network, regardless of which side of the Snort sensor it lies.

The pattern is very regular.
For forty-second intervals spaced about ten minutes apart (sometimes twenty minutes apart),
all five hosts trasmit equal numbers of datagrams
in equal time periods. For example, here are the start and stop times for the last few transmission intervals of New Year's Eve and the first few of New Year's Day:

```
12/31-23:08:10 - 12/31-23:08:51
12/31-23:19:15 - 12/31-23:19:55
12/31-23:30:19 - 12/31-23:30:59
12/31-23:41:23 - 12/31-23:42:03
01/01-00:03:31 - 01/01-00:04:11
01/01-00:14:35 - 01/01-00:15:15
01/01-00:25:39 - 01/01-00:26:19
```

The following excerpt from alerts.030101.gz shows how remarkably synchronized
all five transmissions are:

```
...
01/01-00:04:08.373160  [**] TFTP - External UDP ... internal tftp
server [**]
MY.NET.111.232:69 -> 192.168.0.253:6590
```

```
01/01-00:04:08.373196  [**] TFTP - External UDP ... internal tftp
server [**]
MY.NET.111.230:69 -> 192.168.0.253:6590
01/01-00:04:08.373659  [**] TFTP - External UDP ... internal tftp
server [**]
MY.NET.111.235:69 -> 192.168.0.253:6590
01/01-00:04:08.375042  [**] TFTP - External UDP ... internal tftp
server [**]
MY.NET.111.231:69 -> 192.168.0.253:6590
01/01-00:04:08.377037  [**] TFTP - External UDP ... internal tftp
server [**]
MY.NET.111.219:69 -> 192.168.0.253:6590
01/01-00:04:11.626291  [**] TFTP - External UDP ... internal tftp
server [**]
MY.NET.111.231:69 -> 192.168.0.253:2307
01/01-00:04:11.627899  [**] TFTP - External UDP ... internal tftp
server [**]
MY.NET.111.219:69 -> 192.168.0.253:2307
01/01-00:04:11.629262  [**] TFTP - External UDP ... internal tftp
server [**]
MY.NET.111.232:69 -> 192.168.0.253:2307
01/01-00:04:11.629280  [**] TFTP - External UDP ... internal tftp
server [**]
MY.NET.111.235:69 -> 192.168.0.253:2307
01/01-00:04:11.629296  [**] TFTP - External UDP ... internal tftp
server [**]
MY.NET.111.230:69 -> 192.168.0.253:2307
...
```

Since the ephemeral port used by 192.168.0.253 is the same for all five TFTP server's responses, these could be broadcast amplified packets. In other words, the
stimulus from 192.168.0.253 is a TFTP datagram broadcasted to MY.NET.111.255, and these five hosts are the ones answering this broadcast.

Defensive recommendation:
If, for example,
192.168.0.253 runs your network management software, which periodically tests the stability of 5 network devices using this traffic, then the Snort rule should be modified to eliminate these false positives. But if this traffic is mysterious to you, then these are very high priority alarms. It could be that an adversary has installed a reconnaissance device inside your trusted network, and is taking advantage of allowed RFC-1918 routes inside your network, to systematically gather
data every ten minutes from these five machines.
If that is the case, you should confiscate this
rogue and take steps to shut down any RFC-1918 traffic in your network that is not specifically
authorized by your network planners.

Alerts: Watchlist 000222 IL-ISDNNET-990517 alerts

These alerts seem to be from a custom Snort rule; they are not
part of the default Snort ruleset.
They are triggered by traffic from 212.179.0.0/16, which
RIPE Whois (RIPE) identifies as Bezeq-International's network in
Israel:

```
inetnum:      212.179.0.0  - 212.179.0.255
netname:      REDBACK-EQUIPMENT
mnt-by:       INET-MGR
descr:        BEZEQINT-EQUIPMENT
country:      IL
admin-c:      MR916-RIPE
tech-c:       ZV140-RIPE
status:       ASSIGNED PA
remarks:      please send ABUSE complains to abuse@bezeqint.net
remarks:      INFRA-AW
notify:       hostmaster@bezeqint.net
changed:      hostmaster@bezeqint.net 20021020
source:       RIPE

route:        212.179.0.0/18
descr:        ISDN Net Ltd.
origin:       AS8551
notify:       hostmaster@bezeqint.net
mnt-by:       AS8551-MNT
changed:      hostmaster@bezeqint.net 20020618
source:       RIPE

person:       Miri Roaky
address:      bezeq-international
address:      40 hashacham
address:      petach tikva 49170 Israel
phone:        +972 1 800800110
fax-no:       +972 3 9203033
e-mail:       hostmaster@bezeqint.net
nic-hdl:      MR916-RIPE
changed:      hostmaster@bezeqint.net 20021027
source:       RIPE

person:       Zehavit Vigder
address:      bezeq-international
address:      40 hashacham
address:      petach tikva 49170 Israel
phone:        +972 1 800800110
fax-no:       +972 3 9203033
e-mail:       hostmaster@bezeqint.net
nic-hdl:      ZV140-RIPE
changed:      hostmaster@bezeqint.net 20021027
source:       RIPE
```

The only other clue I could find about the nature of this "Watchlist"
alert is the date 17 May 1999, coded in the alert string as "990517."
This is the day that Ehud Barak became Israel's Prime Minister,
by defeating Benjamin Netanyahu in a public election.
The other number in the alert string could also be a date,
20 Feb. 2000, but I could not connect that date to any event.

Defensive recommendation:
I don't have sufficient information to gauge the nature or severity of these alerts.
I sincerely hope you are not experiencing network abuse motivated by Middle-Eastern political or religious strife.
If you are, I recommend that you retain an expert in international relations or international law, who can recommend appropriate considerations for your network security policy and practice.

Scans and OOS traffic: peer-to-peer (P2P) file sharing

Like most college campuses, your network is brimming with peer-to-peer traffic.
The most popular client is currently Kazaa.
There are 50,488 entries in the scans logs that identify connections to port 1214, the most common port associated with Kazaa. There are thousands more that identify connections with other P2P software, such as Napster (port 6699), WinMX (6699/6257), or Gnutella (6347).
Certainly these scans are just a small subset of your total P2P traffic; these entries are just those connections that happened to meet the threshold of what Snort's spp_portscan preprocessor identifies as a scan.

The protocols used by P2P software such as Kazaa change frequently and without notice, so assessing their impact on the University's security and privacy posture is close to impossible.
Trying to restrict or block P2P traffic is also a losing battle.
The usual reason for the rapid and frequent protocol adjustments is to make the software circumvent security measures and network policing techniques.

Looking through the OOS logs shows that peer-to-peer ports are popular targets of fingerprinting stimuli by tools such as nmap or queso.
There are dozens of Kazaa OOS packets with odd combinations of TCP flags.
But there are others that do not stand out as fingerprinting probes.
Here is an example from Thursday afternoon, which looks like a reasonable packet except for no ACK flag and a zero acknowledgement number:

```
01/02-18:06:59.705859 200.167.121.16:3917 -> MY.NET.150.220:1214
TCP TTL:108 TOS:0x0 ID:30643 IpLen:20 DgmLen:52 DF
```

```
****P*** Seq: 0xB9D2040A  Ack: 0x0  Win: 0x2000  TcpLen: 20
C1 F4 AE 37 14 77 12 F6 C0 04 68 80              ...7.w....h.
```

I suspect that this is not an nmap/queso fingerprinting stimulus,
but instead may be a Kazaa-specific OOS stimulus packet.

For more information about security issues regarding
peer-to-peer file sharing, see the report
Peer-to-Peer File-Sharing Networks: Security Risks (Couch),
which includes a discussion of
the "Benjamin worm" that targeted Kazaa users in 2002.
Defensive recommendation:
Please keep up to date with the network security
news, as more vulnerabilities in P2P networks are certain to surface.

P2P software is used extensively to make unauthorized transmissions
of copyrighted material, contrary to law.
Defensive recommendation:
At the very least,
no user of PU's network should be allowed access unless they
have agreed to your acceptable use policy, which should include
a statement about legal use of (and risks posed by) P2P network
software.

Tools and Processes

Snort-rep and ``Perl Munging''

For Snort alert logs, I have had success with David Schweikert's tool
snort-rep (Schweikert),
so I used it for this analysis.
It expects alerts in either syslog format or Snort's "fast" alert
format,
so I used some scripts to convert the data into "fast" format.
I tried to follow the guidelines in David Cross' excellent book,
Data Munging with Perl. (Cross)

Each alert file is in plaintext format, one line per alert.
When I tried converting these to "fast" format, I found that
some of the alert lines were corrupt.
I wrote the following filter script, 0alertsyntaxbad.plx,

which identified these corrupt lines:

```
#!/usr/bin/perl -w

my @fields;
while (<>) {
  if (/^:/) {
    # many bad alert fragments start with :
    print;
  } elsif (/spp_portscan/) {
    # spp_portscan lines should have two [**] delimited fields
    @fields = split /\s*\[\*\*\]\s*/;
    if ((scalar @fields) != 2) {
      print;
    }
  } else {
    # other alert lines should have three [**] delimited fields
    @fields = split /\s*\[\*\*\]\s*/;
    if ((scalar @fields) != 3 ) {
      print;
    }
  }
}
```

Of the 842,113 alert lines, this filter found 148 which exhibit bad syntax;
I saved those in a "badsyntax.txt" file.
Then I rewrote the script
(essentially just replacing each print with next)
to obtain a complementary filter 1alertsyntaxgood.plx.
It yielded the remaining 841,985 alert lines that were used in the analysis.
I perused the "badsyntax.txt" file after the analysis was complete,
to see if it contained any important event not covered in the analysis.
(It did not, but it did suggest a possible flaw in your Snort system setup
that should be investigated; see the
post-analysis section below for details.)

The following filter, 2alertfastformat.plx,
converts the "good syntax" alert lines into "fast" format:

```
#!/usr/bin/perl -w

while (<>) {
  chomp;
  my @fields = split /\s*\[\*\*\]\s*/;
  my $fieldcount = scalar @fields;
  if ( $fieldcount == 2) {  # spp_portscan format
    print join ' ',
      $fields[0], ' [**]', '[1:0:0]',  # add a zero "sid" field for
snort-rep
      $fields[1], '[**]';
    print "\n";
```

```
    next;
  } elsif ( $fieldcount == 3 ) {  # snort alert format
    print join ' ',
      $fields[0], ' [**]', '[1:0:0]',  # add a zero "sid" field for
snort-rep
      $fields[1], '[**]',
      '[Classification: n/a]',  # snort-rep expects these items too,
      '[Priority: 4]', '{TCP}',  # so insert meaningless versions of
them
      $fields[2];
    print "\n";
    next;
  } else {
    print STDERR "****** Bad field count $fieldcount:";
    print STDERR "";
    print STDERR "\n";
    }
}
```

Finally, snort-rep expects IP addresses as numeric dotted-quads,
but the alert logs are anonymized, with PU's class B numbers
replaced with MY.NET.
I used sed filters to temporarily put PU's hosts in a fake
"99.999.0.0/16" class B.
(In the real world, such IP addresses are invalid,
so snort-rep never mistakes these for non-PU hosts.)

I obtained the snort-rep report by piping all of the assembled tools
together in a shell command line, 3createreport.sh:

```
zcat alert*gz |\
 sed -e 's/MY\.NET/99.999/g' |\
 1alertsyntaxgood.plx |\
 2alertfastformat.plx |\
 snort-rep --source=fast:- --local=99.999.0.0/16 |\
 sed -e 's/99\.999/MY.NET/g' > report.txt
```

The result, "report.txt," laid the foundation for the analysis.
In the analysis above,
Tables 3 (portscans),
5 (alert counts), 6 (alerts on remote hosts),
and 7 (alerts on local hosts)
were essentially cut and pasted from report.txt.
(The full report.txt is almost ten thousand
lines long, and if printed consumes about 160 sheets of paper
at 60 lines per page, so only the most pertinent parts of report.txt
have been copied.)

The snort-rep reports are ideal for importing into a spreadsheet,
because they are plain text and tabular.
For example, to obtain the top five portscan port:host ratios in Table
4,

I imported report.txt's portscan table (Table 3) into StarOffice StarCalc,
added a fourth column that calculates the
ratio of the first two columns, and then sorted the whole sheet
by the results in the fourth column. I also used Microsoft Excel
to sort and study the other alert tables (for example, to sort a
table by IP address and survey alert types generated
by a subnet's hosts).

I also made extensive use of Unix command line tools.
For example,
 I learned in which order the five "Russia dynamo"
streaming sessions occurred, by executing the command pipeline

```
    zcat alert*gz
 | grep Russia | cut -c 71- | grep "^MY" | uniq -c
```

and I learned what time the third session ended via

```
    zcat alert*gz
 | grep Russia | grep "MY\.NET\.105\.204:4930" | tail -1.
```

The link graph illustration was created using
Dia (Dia).

Post-analysis

The file "badsyntax.txt" contained no alerts or patterns
that weren't already addressed in the analysis. But the
bad alert lines themselves showed an interesting pattern.

Here are four example lines from "badsyntax.txt":

12/30-00:12:54.364686 [**] spp_http_decode: IIS Unicode attack
detected [**] MY

```
.NET.71.230:3461 -> 3.155.133.6612/30-00:15:18.160971  [**] NIMDA -
Attempt to e
xecute root from campus host [**] MY.NET.71.230:4795 ->
165.158.251.157:80

:80

01/03-22:30:14.803185  [**] High port 65535 tcp - possible Red Worm -
traffic [*
*] MY.NET.84.15101/03-22:46:28.178412  [**] spp_portscan: portscan
status from M
Y.NET.83.146: 4 connections across 4 hosts: TCP(0), UDP(4) [**]

:65535 -> 172.179.155.153:4366
```

Notice the first two lines. The ":80" fragment completes the first line's "IIS Unicode" alert, at the point where the "NIMDA" alert was inserted mid-line. This is reasonable; such traffic would be aimed at an IIS web server listening on port 80. The same applies to the other pair: the ":65535" fragment completes the "High port 65535" alert at the point where the portscan alert was inserted.

I suspect that there are multiple Snort threads or processes, or perhaps multiple Snort sensors, which occasionally collide during output. To solve this problem, a mechanism needs to be implemented to ensure that each logged alert is an atomic transaction, not interrupted by other alerts. Computer scientists have studied this phenomenon in depth. For more information, consult Andrew S. Tanenbaum's discussion of "Mutual Exclusion" in his chapter on "Synchronization in Distributed Systems." (Tanenbaum, pp 476-482)

Most, but not all "badsyntax.txt" lines can be explained by this observation, so my theory isn't grand and unified. But here is my last
Defensive recommendation:
you should review the Snort code and architecture you are using, to see if my suspicion is accurate. By doing so, you might find a way to clean up your alert logs.

References

APNIC. "Query the APNIC Whois Database."
Asia Pacific Network Information Centre.
URL:http://www.apnic.net/apnic-bin/whois2.pl (29 Jan. 2003)

ARIN. "ARIN WHOIS Database Search."
American Registry for Internet Numbers.
URL:http://whois.arin.net/ (29 Jan. 2003)

Couch, William. "Peer-to-Peer File-Sharing Networks: Security Risks."
SANS Info Sec Reading Room. 8 Sep. 2002.
URL:http://www.sans.org/rr/policy/peer.php (29 Jan. 2003)

Cross, David.
Data Munging with Perl.
Greenwich, CT: Manning Publications Co., 2001.

Dia. "Dia a drawing program." Home page. 1 Jun. 2002.
URL:http://www.lysator.liu.se/~alla/dia/ (29 Jan. 2003)

Microsoft Technet. "Microsoft Security Bulletin MS00-078."
Microsoft Corporation. Originally posted 17 Oct. 2000.
URL:http://www.microsoft.com/technet/security/bulletin/ms00-078.asp
(29 Jan. 2003)

Northcutt, Steven. "Global Incident Analysis Center: Detects
Analyzed 7/29/00." SANS Institute. 29 Jul. 2000.
URL:http://www.sans.org/y2k/072818.htm (29 Jan. 2003)

RIPE
"Query the RIPE Whois Database."
Réseaux IP Européens.
URL:http://www.ripe.net/ripencc/pub-services/db/whois/whois.html (29 Jan. 2003)

SANS[1]. "SANS FLASH: New Trojan Sending Data To Russia."
Neohapsis Archives. 28 Jul. 2000.
URL:http://archives.neohapsis.com/archives/sans/2000/0068.html (29 Jan. 2003)

SANS[2], Fearnow, Matt and Stearns, William.
"Adore Worm." SANS Instititue. 12 Apr. 2001.
URL:http://www.sans.org/y2k/adore.htm (29 Jan. 2003)

Schweikert, David. "snort-rep - a Snort Reporting Tool."
URL:http://people.ee.ethz.ch/~dws/software/snort-rep/ (29 Jan. 2003)

SecurityFocus Online. "Microsoft IIS and PWS Extended Unicode Directory Traversal Vulnerability." Bugtraq id 1806. Updated 10 Sep. 2001.
URL:http://online.securityfocus.com/bid/1806 (29 Jan. 2003)

Sophos[1]. "Sophos virus analysis: W32/CodeBlue."
URL:http://www.sophos.com/virusinfo/analyses/w32codeblue.html
(29 Jan. 2003)

Sophos[2]. "Sophos virus analysis: W32/Nimda-D."
URL:http://www.sophos.com/virusinfo/analyses/w32nimdad.html
(29 Jan. 2003)

Sophos[3]. "Sophos virus analysis: Troj/CodeRed-II."
URL:http://www.sophos.com/virusinfo/analyses/w32codered2.html
(29 Jan. 2003)

Sophos[4]. "Sophos virus analysis: Linux/Adore."
URL:http://www.sophos.com/virusinfo/analyses/linuxadore.html
(29 Jan. 2003)

Tanenbaum, Andrew. Modern Operating Systems.
Englewood Cliffs, NJ: Prentice Hall, 1992.

Turkia, Mike. "SANS Intrusion Detection Practical." GIAC. 28 Jan. 2001.
URL:http://www.giac.org/practical/Miika_Turkia_GCIA.html (29 Jan. 2003)