Global Information Assurance Certification Paper

“Snort Overdrive”

GCIA Practical Assignment, version 3.4
(Revised September 23, 2003)

Abstract

The paper was written during Spring 2004. Is divided into three parts, and follows the guidelines of Assignment 3.4, which is valid until August 2004.

In part one, some of the issues with encryption (in relevance to network intrusion detection) are discussed. A few of the more commonly available encryption techniques are analysed to see if Snort rules can be written to detect them (or the abscense of them where they are expected).

In part two, three network detects are analysed.

Detect one is traffic with loose source routing. The raw data comes from incidents.org

Detect two is buffer overflow attempt targeting the Microsoft Workstation service (described in MS03-049)

The final detect analyses the behaviour of the DeadHat.B worm. While most antivirus companies already have descriptions of it, the author found that they generally do not describe a worm's network traffic to the extent possible. This was true also for DeadHat.B.

Part three is an analysis on five days worth of logged data from an university in the United States. The dates analysed was April 7 to April 11.

The analysis in part three was performed with a MySQL database. The raw data had some corrupt lines in it, so some effort was spent removing those. Findings include several false alerts, but also signs of compromised internal hosts (mostly by worms).

Submission by Patrik Sternudd,
on June 25, 2004

Table of Contents

1. Part 1 – NIDS and dealing with encryption

1.1. Introduction

Sometimes, I get the feeling the attitude towards NIDS (Network Intrusion Detection Systems) and encrypted data is that as soon as you encounter it, you are out of luck. But is that really the case? Should we just shrug and say *"there's nothing for it"* and go for a coffee? Personally, I do not think so.

In this paper, I will take a look on some common methods for encrypting network data and attempt to answer two primary questions:

1. How do we know we are dealing with encryption?
2. Can we write rules for Snort [0], allowing us to detect it?

My goal is to show that even though most of the data is encrypted, there is still some information available which we can use to enhance our security.

In order to limit the scope, there are some issues I will not cover:

- Loki and other innovate backdoor tunnelling which may or may not be encrypted.
- Strategic placement of NIDS sensors to sidestep the problem (it can be argued this does not help – what if encrypted traffic appears nevertheless?).

Neither will I attempt to cover all possible ways to do encryption; I will only select a few to show that it is, in fact, possible to detect this kind of data when desirable.

This paper is highly technical. The reader should be familiar with TCP/IP concepts and terminology. Generally accepted acronyms may be used without further explanations. A basic understanding of cryptography may be useful, but is not required. Familiarity with Snort is likely to be helpful.

All IP addresses and host names has been obfuscated; any semblance with real networks or domains are entirely coincidental, and should be regarded as such. Network traces has been arbitrarily truncated to allow for easier reading.

1.2. Why should we care at all?

As security professionals, why should we care? Is not data encryption a good thing, something that adds security to our network?

Well, it is, but only when we are in control of it. Encryption could be used against us as well. It is referred to within that context in several papers dealing with NIDS evasion techniques.

It could also be used in various backdoors and worms calling home.

Another thing to worry about is if - and, in particular, how - it is utilised internally. A lot of encrypted sessions on our internal networks could make us lose the real picture of what is going on. This is particularly true if end users are bringing up encrypted tunnels to the outside. On the other hand, we will certainly want to encrypt our sessions to critical systems such as IDS sensors. System administrators wants (or at least should want) to use SSH to manage servers and network devices. And end users want the privacy offered by encryption services when they access certain resources on the Internet (such as on-line banking).

I will further make three observations that together provide the rationale for the focus of this paper (which is aimed more towards detecting policy violations than the latest worm or hacker attacks):

1. In my experience, people sometimes seem to feel that the IANA[1] assigned numbers[2] are as law. If it comes on port 22, it must be SSH, because it says so, right there in the specification!

2. On a similar note, there are a few ports that almost always seem to be open outbound, even though the firewall is rather restrictive otherwise. Among these, TCP/80 (HTTP) and TCP/443 (HTTPS) are by far the most frequent ones.

3. Security personnel would do well never to underestimate the creativity of end users who feel they are impeded by the firewall or other security measures. They will find ways around it, sooner or later (this is increasingly true for technically savvy people, such as normal system administrators, or simply just people fooling around with computers at home).

The corollary: seeing TCP traffic on ports 22 and 443, we might assume we are seeing SSH and HTTPS. That may unfortunately be an incorrect assumption. What if someone ran a SSH server on port 443 in order to sneak through the firewall?

The issues described above could of course be mitigated by having restrictive policies and firewall rulesets in place, and only allowing traffic to pass through application layer proxies. But still, many sites do have quite liberal rulebases (or no firewalls at all, e.g. some universities). In those cases, detecting possibly malicious traffic becomes even more important.

1.3. What should we look for, then?

While one could theoretically perform some sort of analysis on the encrypted payload in order to pinpoint encryption algorithms, it does not work out very well in reality. Not yet, at least, although anomaly[3] detection systems may be of some use (they do not necessarily need to know the specifics of the payload; instead they may

1 Internet Assigned Numbers Authority
2 As of Jan 2002, the Internet Assigned Numbers is no longer a RFC[1]; the latest version is instead available on the IANA website[2].
3 For more information regarding different types of intrusion detection systems, I recommend the book “Intrusion Detection” by Rebecca Gurley Bace[3].

alert on the fact that previously unknown data is flowing from an usually quiet internal system to the Internet).

For one thing, there are numerous algorithms that are commonly used. Then consider that many of these may be run in different modes (for example, SSL v3.0 supports 37 different cipher specifications). It would take an extensive amount of work (and a vast competence in cryptography, which I do not have) to be able to detect all conceivable combinations.

Add to this the fact that this method will require more data to perform the analysis, and we have reached a point where we cannot automate the process without spending huge memory and CPU resources (and perhaps not even then). The old but still excellent treatise by Thomas H. Ptacek and Timothy Newham[4] mentions this in passing.

I will take another approach and take a look at the lower levels of the communication which are supporting the different algorithms. Perhaps we could call them "encryption carriers". The most common of these are SSL, TLS, SSH and IPSec (there are quite a few of them, the list could definitely be larger).

If we can detect the carriers in a somewhat simple manner, it does not really matter that we do not know the exact algorithm. And unless we want to ensure that we are using secure (in regard to acceptable risk) methods, or want to decrypt the data, we do not even care. If I see a SSL Handshake on the wire, I can then assume that some encryption is going on. The same thing applies for IPSec SA negotiations.

For the remainder of this paper, I will take a closer look at SSL, TLS and SSH in order to describe their characteristics. These characteristics will later be used to write rules for detection. I had originally included an IPSec part as well, but it was removed to make the other sections more in-depth. IPSec is also easier to control at the border – block IP protocols 50, 51 and UDP port 500 and you will probably be safe[4].

1.4. A problem (or two) with Snort

I found that for many detects, it would be useful to look only at specific packets in a TCP session[5]. But how can we tell Snort we are only interested in the first packet after a three-way-handshake, or TWH (this would normally be packet number four in a TCP session)? In theory, it is not has hard as it initially appears; such packets have relative SEQ (sequence) and ACK (acknowledgement) numbers of 1 and a payload greater than zero (if it is zero, it is the ACK completing the handshake).

The problem is that Snort has no capability to inspect *relative* ACK or SEQ numbers, only absolute ones (it is of course much harder to deal with relative ones, as this require awareness of the session state).

4 It is always possible to tunnel everything through something else, butmost VPN services (even those that are encapsulated to handle NAT) use IKE on UDP 500.

5 TCP is described in RFC 793 [5]

To my everlasting gratitude, during the time I considered how to resolve this problem, the Snort Team released Snort 2.1.1 with a new capability called *flowbits*[6], allowing user defined attributes to be set on flows (I do have a hunch that I am not using them quite as the authors imagined, but nevertheless, it solved my problem).

While proceeding with the rule creation, I found another problem. The stream4 preprocessor tags a stream as established as soon the TWH is completed. Well and good, but it then passes it on to the detection module. So, when using the established keyword without any content checks, the first packet it will trigger on is *not* the first data packet after the TWH, but the ACK message completing the TWH. That really messed up my rules.

One workaround would be to use the "flags: A+" check, but getting rid of that was one of the reasons for making the "established" check. Also, even though they are extremely common, PSH (push) and other TCP flags are optional, not required. The workaround I finally decided for was to check for data sizes greater than zero (the ACK completing the TWH should not contain any data).

1.5. SSH

SSH connections are quite straightforward to detect. Section 4.2 (Protocol Version Exchange) in the Internet-Draft[7] states:

> *When the connection has been established, both sides MUST send an identification string. This identification string MUST be*
>
> *SSH-protoversion-softwareversion SP comments CR LF*

As a quick note: the draft also says this string not necessarily must be the first line sent, but it appears to be the case in most implementations.

All we need to do is to take a look at the first packet after the completed TWH. If the first four octets contain the ASCII string "SSH-", chances are good that we are seeing a SSH session. A sample packet is showed in figure 1. The TWH is not included in the sample, nor will it be in subsequent ones.

```
IP server.net.22 > client.org.1536: P 1:21(20) ack 1

0x0000   4500 003c 946c 4000 3906 XXXX XXXX XXXX        E..<.l@.9.......
0x0010   XXXX XXXX 0016 0600 4dfb 7c0c fca7 1442        ........M.|....B
0x0020   5018 c4e0 XXXX 0000 5353 482d 322e 302d        P.......SSH-2.0-
0x0030   5375 6e5f 5353 485f 312e 300a                  Sun_SSH_1.0.
```

Fig 1 SSH Server response.

6 This is covered in the Snort documentation which is available online [6].
7 Ylonen & Lonvick p.4. [7]

A snort rule to detect it can be written thus:

```
alert tcp any 22 -> any any (msg: "SSH server response"; \
            flow:established; \
            flowbits:isnotset,FIRST_PACKET_SEEN; \
            content: "SSH-"; depth:4; \
            flowbits:set,FIRST_PACKET_SEEN; )
```

Note that the flowbits label "`FIRST_PACKET_SEEN`" is entirely arbitrary (it is a convenient one in my opinion, but I am biased). So what does it do? Well, first it checks that the flow is in an established state. Then it proceeds to check that the attribute "`FIRST_PACKET_SEEN`" is not set (if it is, this flow has been dealt with before). The actual content check follows, and if all condition match, the "`FIRST_PACKET_SEEN`" attribute is added to the flow.

In order to detect SSH going on non-standard ports, the port number 22 should of course be prefixed with an exclamation mark.

1.6. SSL and TLS

A quick primer on SSL and TLS has been written by Holly Lynne McKinley as a GSEC practical [8], and is recommended reading for those without prior knowledge of those protocols. For the purpose of this document, the reader should be aware that there are different internal protocols. At the lowest level is the Record Protocol. Go up one level and we find the Handshake Protocol, which is responsible for setting up new sessions. For doing the session initiation, Hello messages are used (the data structures differ between the variants).

> *When a client first connects to a server it is required to send the client hello as its first message.*[8]

Great! Just what we needed. In the next three sections I will go through these variants and show the structures they are using (the specifications are not really quotation-friendly, so I will use my own style).

Again, (as with SSH), it is the fourth packet in the TCP session that is of interest. Please be aware of the difference between SSH, though: In SSL/TLS, it is the client that will send the first data packet, not the server.

Note that I have written the version as a 16 bit hexadecimal number. In the specifications, it is actually composed of two separate octets, the first being the major number and the second the minor.

SSL version 2.0

A SSL version 2.0 [10] (I will henceforth refer to it as SSL v2) header is structured in the following way (only interesting portions are showed):

8 Dierks & Allen, p.33. [9]

SSL version 2.0 header			
Byte offset	**Field**	**Value**	**Meaning**
0,1	Record Length	<variable>	
2	Message Type	0x01	Client Hello
3,4	Client Version	0x0002	SSL v2.0
5,6	Cipher Specification Length	<variable>	
9..	Cipher Specifications	<variable>	

A sample packet is showed in figure 2 (below).

```
IP client.org.1554 > server.net.443: P 1:46(45) ack 1

0x0000   4500 0055 4ec1 4000 8006 XXXX XXXX XXXX        E..UN.@.........
0x0010   XXXX XXXX 0612 01bb 024e 7541 45c7 9ee3        .........NuAE...
0x0020   5018 fc00 XXXX 0000 802b 0100 0200 1200        P........+......
0x0030   0000 1001 0080 0300 8007 00c0 0600 4002        ..............@.
0x0040   0080 0400 80a4 46ea 981a 400d b785 11c0        ......F...@.....
0x0050   188b 0b9c bf                                  .....
```

Fig 2 SSL v2 Client Hello.

The bytes 5 and 6 (from the beginning of the payload) tells us that the cipher specifications takes up 18 bytes. Each specification consists of three octets of data (for example, "`01 00 80`" equates to "`SSL_CK_RC4_128_WITH_MD5`"), which means that we should have 6 different specifications in this message. At offset 9 and 18 bytes onward, the various specifications included in the message are defined. In version 2, all valid alternatives have the middle octet set to zero. A quick comparison against that format shows the above packet to be correct in this regard. Unfortunately, we are unable to use the specifications for our rule writing, as they are variable (the reason this section is here will become apparent later).

Sadly, the string "`01 00 02`" in that particular position is not too much to go on. We need to make our signature more precise somehow, or we might drown in false positives. This can be achieved by reading through the specification and calculate the maximum payload size. A specification compliant layer two Client Hello will have a maximum payload of 80 (for this to happen, all possible cipher specifications would have to be enabled, which is not very likely).

The snort rule could go along these lines:

```
alert tcp any any -> any !443 (msg: "SSL v2 client hello"; \
            flow:established; \
            dsize: < 81; \
            flowbits:isnotset,FIRST_PACKET_SEEN; \
            content: "|01 00 02|"; offset:2; depth:3; \
            flowbits:set,FIRST_PACKET_SEEN; )
```

SSL version 3.0 and TLS version 1.0

SSL version 3.0 [11] and TLS version 1.0 [9] (henceforth referred to as SSL v3 and TLS) both use the same header. The only difference is the version numbers (which is `0x0300` and `0x0301`, respectively).

SSL version 3.0 and TLS 1.0 header			
Byte Offset	***Field***	***Value***	***Meaning***
0	Record Content Type	0x16	Handshake
1,2	Protocol Version	0x0300	SSL v3.0
3,4	Record Length (bytes)	<variable>	
5	Handshake Type	0x01	Client Hello
6,7,8	Message Length (bytes)	<variable>	
9,10	Client Version	0x0300	SSL v3.0

The obligatory packet dump (figure 3):

```
IP client.org.1553 > server.net.443: P 1:121(120) ack 1

0x0000   4500 00a0 4e70 4000 8006 XXXX XXXX XXXX        E...Np@.........
0x0010   XXXX XXXX 0611 01bb 010f 5aa5 4114 af17        ..........Z.A...
0x0020   5018 fc00 XXXX 0000 1603 0000 7301 0000        P...........s...
0x0030   6f03 0000 009a bc55 cacd 611f 5f93 2919        o......U..a._.).
0x0040   4768 87ba 1ac5 4cd2 5b02 4992 c0a1 34aa        Gh....L.[.I...4.
0x0050   8c65 1720 47cb a5e4 5bd0 72ed b3f4 a069        .e..G...[.r....i
0x0060   d719 15e2 2a14 c697 65fa 9d30 d567 3cd2        ....*...e..0.g<.
0x0070   afaf 552c 0028 0039 0038 0035 0033 0032        ..U,.(.9.8.5.3.2
0x0080   0004 0005 002f 0016 0013 feff 000a 0015        ...../..........
0x0090   0012 fefe 0009 0064 0062 0003 0006 0100        .......d.b......
```

Fig 3 SSL v3 Client Hello.

As can be seen, this header is more complex than the one used by the previous version. This requires a more complex snort rule, but with the advantage that we are reducing the likelihood of false positives. Only the SSL v3 sample rule is displayed.

```
alert tcp any any -> any !443 (msg: "SSL v3 client hello"; \
            flow:established; \
            flowbits:isnotset,FIRST_PACKET_SEEN; \
            content: "|16 03 00|"; depth:3; \
            content: "|01|"; offset:5; depth:1; \
            content: "|03 00|"; offset:9; depth:2; \
            flowbits:set,FIRST_PACKET_SEEN; )
```

SSLv3 or TLS 1.0 in backward compatibility mode

Both SSL v3 and TLS can be backwards compatible with SSL v2 This is done by using a SSL v2 Client Hello message with the version number of `0x0300` or

0x0301, respectively. The higher number of possible cipher specifications means the maximum payload will be 170 or 161 bytes (TLS has fewer specifications than SSL v3).

A modified snort rule could look thus (this time, the TLS sample is showed):

```
alert tcp any any -> any !443 (msg: "TLS backwards compatible ClientHello"; \
            dsize: < 162; \
            flow:established; \
            flowbits:isnotset,FIRST_PACKET_SEEN; \
            content: "|01 03 01|"; offset:2; depth:3; \
            flowbits:set,FIRST_PACKET_SEEN; )
```

SSL v3 would of course use “dsize: < 171” instead of 162.

1.7. Running a Snort Ruleset

My theories naturally had to be put to the test. A sniffer running snort was put on a network segment just between the corporate firewall (at an undisclosed site) and the upstream router. The ruleset used is described in the next section (for brevity, the complete ruleset is not included; however, the rules presented in the previous sections are used without other alterations than different port numbers or rule actions).

The ruleset

First, in order to use pass rules to ignore traffic we know is good (or rather do not care about in this context), it is necessary to reconfigure the action order:

```
config order: pass alert log activation dynamic
```

Then, of course, we need the stateful capability and the preprocessors that use it:

```
config stateful
preprocessor flow
preprocessor stream4
```

As for the rules, it begins with a rule for alerting on SSH traffic coming on non-standard ports (i.e. anything except port 22). A rule for SSH on the standard port is also active, mostly as a reference.

Then there are three rules to alert on SSL versions 2 and 3, plus TLS. Two additional ones handle backward compatible Client Hello's for TLS and SSL v3. These five rules only trigger on ports different from 443.

The following five rules are identical to the previous ones except for the facts they are monitoring port 443 only, and they are passing the traffic instead of alerting (there was a lot of encrypted web traffic on that port, which had to be excluded).

The final rule (shown below) is the reason for the changed config order. It will alert on anything on port 443 that has not been passed by the previous rules. Without

being able to lock it down to the first packet after TWH, this rule would have generated too many false positives to have been even remotely useful.

```
alert tcp any any -> any 443 (msg: "Non TLS/SSL on TLS/SSL port"; \
                      !dsize:0; \
                      flow:established,to_server; \
                      flowbits:isnotset,FIRST_PACKET_SEEN; \
                      flowbits:set,FIRST_PACKET_SEEN; )
```

Some SSL v2 false positives

As I had feared, the static content of the SSL v2 header was not specific enough, the pattern may occur in non-SSL traffic as well.

```
[**] [1:0:0] SSL v2 client hello [**]
03/06-13:00:33.849067 web server.org:80 -> client.com:1432
TCP TTL:254 TOS:0x0 ID:6050 IpLen:20 DgmLen:116 DF
***A**** Seq: 0xAC50B003  Ack: 0x3BE76F  Win: 0x2398  TcpLen: 20
```

The packet triggering the alert is displayed in figure 4.

```
IP web server.org.80 > client.com.1432: . 2890969091:2890969167(76) ack 3925871
win 9112 (DF)

0x0000   4500 0074 17a2 4000 fe06 XXXX XXXX XXXX        E..t..@.........
0x0010   XXXX XXXX 0050 0598 ac50 b003 003b e76f        .....P...P...;.o
0x0020   5010 2398 XXXX 0000 0d61 0100 0211 0304        P.#......a......
0x0030   2112 3105 4151 6113 2271 8132 0614 91a1        !.1.AQa."q.2....
0x0040   b142 2324 1552 6233 34c1 7282 4307 2592        .B#$.Rb34.r.C.%.
0x0050   0853 d1f0 6373 3516 e1a2 f1b2 8326 4493        .S..cs5......&D.
0x0060   5464 45c2 a374 3617 18d2 55e2 65f2 b384        TdE..t6...U.e...
0x0070   c3d3 75e3                                      ..u.
```

Fig 4 False SSLv2 packet.

Well, that packet does not really look like a SSL v2 header. I see no trace of any cipher specifications, and the payload is larger than usual for SSLv2 packets (of course, knowing that web server.org is a legitimate web server also helps the analysis, but the most telling fact is undoubtedly the absence of the cipher specifications). A couple more of these appears in the alert log, but they are not too frequent, just a minor annoyance.

SSH traffic

A bunch of alerts came within a very short interval (a subset being showed below):

```
[**] [1:0:0] SSH server response [**]
03/08-04:54:44.763433 172.16.1.222:22 -> obvious.scanner:38711

[**] [1:0:0] SSH server response [**]
03/08-04:54:44.815555 172.16.1.231:22 -> obvious.scanner:38720

[**] [1:0:0] SSH server response [**]
03/08-04:54:44.822176 172.16.1.232:22 -> obvious.scanner:38721

[**] [1:0:0] SSH server response [**]
03/08-04:54:44.826248 172.16.1.242:22 -> obvious.scanner:38731
```

Nothing very exciting, seems to be a normal portscan targeting systems at the corporate network. We do have the TWH (it must be completed for this rule to trigger), so it is probably not spoofed. From looking at the source port numbers of the scanner (they are increasing over time), I would hazard a guess that a common TCP Connect scan is being done, targeting port 22 only. Notice that the complete network is being scanned, with destination IP incrementing sequentially. Not too clever, and easy to detect (hey, that signature was not even built for detecting port scans!).

There is an extra bonus, though. While looking at the payload with tools as tcpdump or Ethereal, we can see the SSH server version right after the initial “SSH-” string (please refer to figure 1 for a sample). The four responses yielded two different versions of OpenSSH: 3.4p1 and 3.7.1p2. Why is this interesting? Well, according to the security page on the OpenSSH website [12], all versions prior to 3.7.1p2 has one or more security vulnerabilities. In other words, we have vulnerable remote access protocols running at “our” site, and all inhabitants on the Internet seems to be free to connect to them at will. Not very cool at all.

The next SSH-related alert looked like this:

```
[**] [1:0:0] SSH server response on non-standard port [**]
03/08-09:41:18.394286 large_netblock.isp:2214 -> client.org:1266
TCP TTL:51 TOS:0x0 ID:43435 IpLen:20 DgmLen:65 DF
***AP*** Seq: 0xAF306541  Ack: 0xB6FD07FA  Win: 0x16D0  TcpLen: 20
```

Obviously, someone is running SSH from the internal network to an external SSH server on the Internet, and it is listening on port 2214. That might be worth looking into. It could be a compromise, but it would not surprise me if it is someone connecting to their home computer during work hours.

Something sneaky

Snort picked up the following.

```
[**] [1:0:0] Non TLS/SSL on TLS/SSL port [**]
03/08-11:00:38.541282 client.com:52374 -> server.net:443
TCP TTL:115 TOS:0x0 ID:13497 IpLen:20 DgmLen:46 DF
***AP*** Seq: 0xDA5B044C  Ack: 0xF82BD099  Win: 0xFBFA  TcpLen: 20
```

The offending packet is showed in figure 5.

```
IP client.com.52374 > server.net.443: P 3663397964:3663397970(6) ack 4163621017
win 64506 (DF)

0x0000   4500 002e 34b9 4000 7306 5891 XXXX XXXX        E...4.@.s.......
0x0010   XXXX XXXX cc96 01bb da5b 044c f82b d099        .........[.L.+..
0x0020   5018 fbfa XXXX 0000 7f7f 4943 4100             P.........ICA.
```

Fig 5 Suspicious packet on port 443 (interesting parts highlighted in red)

Now, that packet really should be SSL or TLS, but obviously it is not. More of these packets are seen throughout the capture (which spanned over a couple of days). They do not occur frequently, just a few times for each source host and day (more sessions were seen from other sources but to the same destination). The payload did always consist of the hexadecimal sequence visible in the capture ("`7f7f 4943 4100`"), so it can probably be used for a new rule to detect this kind of traffic.

The string "`ICA`" corresponding to the hexadecimal value of "`49 43 41`" is a good clue. MetaFrame [13] is quite a nice remote desktop product from Citrix, and the protocol they use is called ICA (Independent Computing Architecture). The port usually associated with this traffic is 1494 (it is registered with IANA), but I would guess there is some kind of firewall-evading business going on here.

1.8. Conclusion

Encrypted network traffic is a challenge for intrusion detection and should be deployed only in a controlled manner. This is of course a policy issue more than anything else, but if controls are in place, it becomes much easier decide whether encrypted data is acceptable or not ("hey, there should be no SSL going coming from that network!"). It does not help if we can detect SSL unless we also can decide whether the traffic is legitimate or not.

Even though only a specific service is expected (as listed by IANA) on a certain port, there is no guarantee someone does not deploy another service through it. An encrypted service may be used on a port not usually associated with encrypted data or vice versa. A NIDS such as Snort can be used to detect such anomalies.

The “normal” (if there is such a thing) usage for Snort is being a signature based NIDS, or possibly a sniffer. The purpose in these cases is to detect specific attacks. However, if a signature can be written for a protocol, Snort can also easily be used to detect that protocol on unusual ports. It can of course also be used to detect anything except the, for a specific port, expected protocols. This is perhaps more of a network policy appliance than an outright intrusion detection one.

For the really sneaky and patient attacker, the discussed measures will not be enough. If someone did the extreme and tunnelled TCP over HTTP over TLS on port 443, none would be the wiser.

While it is certainly possible to write Snort rules to detect encryption carriers, it is a bit awkward. A specific crypto preprocessor would be really welcome in the future (it would probably be a lot faster than the rules I wrote). Another thing I think Snort could benefit from is the possibility of more complex rules (I would absolutely love to be able to use logic operators). This would eliminate the need for the “pass” rules I used to detect anything except the known good traffic. Oh, and while I am at it. Pure not rules would be even nicer (i.e. to prepend an exclamation mark before the content check to match anything except the specified content).

In his GSEC practical about IDS evasion, Corbin Del Carlo writes:

> *To protect against an attacker encrypting their commands, future NIDSs should alert if they see any outbound encrypted session from any host that does not normally conduct encrypted sessions, as well as any large number of inbound session initiations that would be typical of a brute force password guessing attack over SSH or an SSL VPN.*[9]

A crypto preprocessor could possibly make anomaly detection engines such as SPADE[10] more accurate, if they could be made to cooperate.

To conclude my findings, I would say that it is indeed possible to detect encrypted traffic by looking at the carrying protocols. It is my assumption that other carriers can be identified and put into rules by the same method I used (i.e. by reading the specifications). And as usual, a human mind is still required to tell false positives from real events. There are still no replacements for trained analysts.

9 Del Carlo, p.6. [14]
10 SPADE is a Snort Preprocessor – Statistical Packet Anomaly Detection Engine [15]

1.9. References

[0] ”Snort”. URL: http://www.snort.org/ (25 Jun 2004)

[1] Reynolds, Joyce K (editor). ”Assigned Numbers: RFC 1700 is replaced by an On-Line Database”. Jan 2002. URL: http://www.ietf.org/rfc/rfc3232.txt (25 Jun 2004)

[2] IANA. ”TCP and UDP Port Numbers” 19 Mar 2004. URL: http://www.iana.org/assignments/port-numbers (25 Jun 2004)

[3] Bace, Rebecca Gurley. ”Intrusion Detection”. Macmillan Technical Publishing. 2000. ISBN 1-57870-185-6

[4] Ptacek, Thomas H & Newsham, Timothy N. ”Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection”. Jan 1998. URL: http://www.snort.org/docs/idspaper/ (25 Jun 2004)

[5] Postel, Jon (editor). ”Transmission Control Protocol”. Sep 1981. URL: http://www.ietf.org/rfc/rfc0793.txt (25 Jun 2004)

[6] The Snort Project. ”SnortUsers Manual” 12 May 2004. URL: http://www.snort.org/docs/snort_manual.pdf (24 Jun 2004)

[7] Ylonen, Tatu & Lonvick, C (editor). ”SSH Transport Layer Protocol”. Jun 2004. URL: http://www.ietf.org/internet-drafts/draft-ietf-secsh-transport-18.txt (25 Jun 2004)

[8] McKinley, Holly Lynne. “SSL and TLS: A beginner's guide”. 2003. URL: http://www.sans.org/rr/papers/44/1029.pdf (25 Jun 2004)

[9] Dierks, T & Allen, C. ”The TLS Protocol Version 1.0”. Jan 1999. URL: http://www.ietf.org/rfc/rfc2246.txt (25 Jun 2004)

[10] Hickman, Kipp E.B. ”SSL 2.0 PROTOCOL SPECIFICATION”. 9 Feb 1995. URL: http://wp.netscape.com/eng/security/SSL_2.html (25 Jun 2004)

[11] Freier, Alan O. et al ”The SSL Protocol Version 3.0”. 18 Nov 1996. URL: http://wp.netscape.com/eng/ssl3/draft302.txt (25 Jun 2004)

[12] ”OpenSSH Security”. URL: http://www.openssh.org/security.html (25 Jun 2004)

[13] ”Citrix Remote Office Connectivity Solutions for the On-Demand Enterprise”. URL: http://www.citrix.com/site/PS/solutions/solution.asp?solutionID=1408 (25 Jun 2004)

[14] Del Carlo, Corbin. ”Intrusion detection evasion: How Attackers get past the burglar alarm”. 25 Sep 2003. URL: http://www.sans.org/rr/papers/index.php?id=1284 (25 Jun 2004)

[15] Hoagland, James & Staniford, Stuart. “SPICE / SPADE”. URL: http://www.silicondefense.com/software/spice/index.htm (23 Mar 2004)

2. Part 2 – Network detects

I will attempt to explain most issues I write about. However, I do expect the reader to have at least a basic knowledge of Snort [0] and tcpdump output. I will explain why rules trigger, but I will not explain every field. Similarly, I may truncate the tcpdump or Snort output in an arbitrary manner. I expect the reader to have enough knowledge about TCP/IP, Snort and tcpdump to understand what they are seeing.

2.1. Detect 1 (incidents.org) - LSRR attempt

Source of trace

The detect described in this section was found in the file 2003.12.15.7, which is one of several in the archive listed below:

http://www.incidents.org/logs/Raw/2003.12.15.tgz [1]

From looking at MAC and IP addresses and applying techniques described in Ian Martin's GCIA practical [2], I can speculate a bit about the network layout.

It is known that the addresses are obfuscated. I will make the assumption that subnet relationships has been preserved. If not, all bets are off.

All IP addresses on 10.10.10.0/24 have different MAC addresses (see figure 6), so it feels logical to assume this is the segment where the Snort sensor is located.

```
$ tcpdump -enqr 2003.12.15.7 "ip src net 10.10.10.0/24" | cut -d " " -f 2,9 \
    | cut -d "." -f 1-4 | sort | uniq -c

      4 00:0c:29:14:1e:63 10.10.10.142
   2257 00:0c:29:9e:ef:53 10.10.10.224
     21 00:50:56:40:00:64 10.10.10.2
     33 00:50:56:40:00:6d 10.10.10.1
    260 00:a0:c9:ba:6d:85 10.10.10.196
   2338 00:d0:59:c6:5e:14 10.10.10.141
```

Fig 6 Excerpt of local MAC addresses. Field 2 is SRC MAC, field 9 is SRC IP.

All addresses on other internal networks appears to have the same MAC address – 00:50:56:40:00:6d – which probably means it is the gateway to those networks.

```
$ tcpdump -enqr 2003.12.15.7 "ip and not dst net 10.10.10.0/24" \
    | cut -d " " -f 4,11 | cut -d "." -f 1-4 | sort | uniq -c

   2359 00:50:56:40:00:6d, 172.20.11.2
   9708 00:50:56:40:00:6d, 172.20.11.80
     29 00:50:56:40:00:6d, 192.168.17.1
   3877 00:50:56:40:00:6d, 192.168.17.135
```

Fig 7 Excerpt of destionation MAC addresses. Field 4 is DST MAC, field 11 is DST IP.

Returning to figure 6, the IP address associated with MAC 00:50:56:40:00:6d can be seen. The address, 10.10.10.1, is a likely IP for a gateway. Experience says it should be a router, switch or firewall. Can the device type be deduced?

According to the IEEE database [3], the OUI[11] "00:50:56" belongs to VMWare, Inc.

That is somewhat interesting. Last time I checked, VMWare was not one of the major router vendors. If I was to speculate, I would say it is a Linux instance (in VMWare) running routing software (it does of course not have to be VMWare; MAC addresses may be spoofed by any decent OS, so looking at the IEEE OUI is merely an indication, not a proof). Another indication supporting my theory is that the MAC address of 10.10.10.2 is very similar (the last number is 0x64 instead of 0x6d); I assume it is the same system.

OK, The first hop is covered. Are there more of them? I cannot be sure of course, but there are a lot of returning packets from the target. All said packets have a TTL of 62. Assuming the initial TTL is 64 (which is a common enough value), and that 10.10.10.1 is doing its job (i.e. decreasing the value by one before passing it to the final destination), it follows there must be at least another routing device in between. I will qualify that guess by saying that it is more likely a router than a firewall, since I see the responses (RST/ACK) from a complete portscan coming back ("wide open" is just not the most common firewall configuration I have come across so far – if the filtering capability is not required, a router will usually be preferred since it is faster).

With the information available at this point, I expect the network to be designed along the lines shown in figure 8 (at least the parts of it that is relevant to the detect).

11 Organisation Unique Identifier [4]

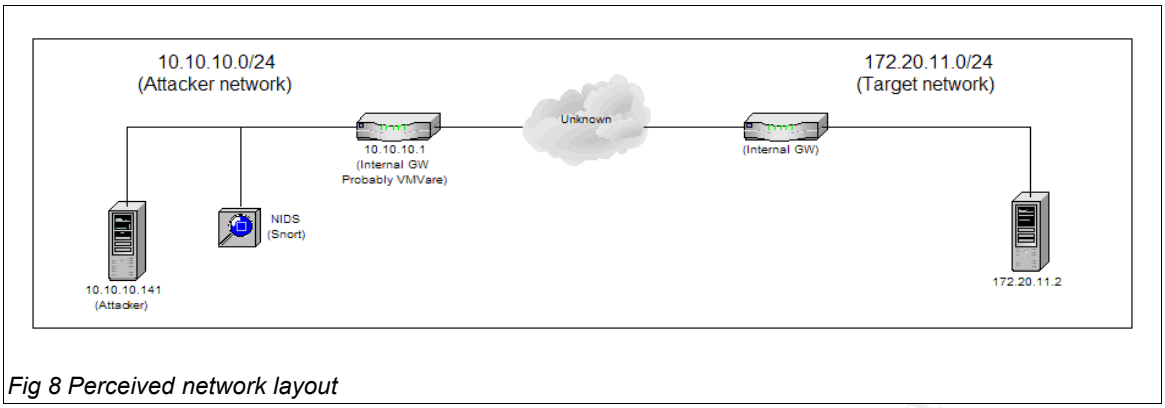


Fig 8 Perceived network layout

Detect was generated by

The README[12] file clearly states that the traces were recorded by a Snort ruleset of unknown version. Therefore it made sense to run the binary logfile through another instance of Snort to see just what was going on.

For this purpose, I used Snort version 2.1.1 (build 24) with the standard ruleset. The following alerts (of same type) caught my interest (reference fields are removed from alerts two and three):

```
[**] [1:500:2] MISC source route lssr [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/18-21:09:27.593273 10.10.10.141:35512 -> 172.20.11.2:22
TCP TTL:64 TOS:0x0 ID:31113 IpLen:24 DgmLen:44
IP Options (1) => LSRR
******S* Seq: 0x68474C99  Ack: 0x0  Win: 0x200  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS418]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0909]
[Xref => http://www.securityfocus.com/bid/646]

[**] [1:500:2] MISC source route lssr [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/18-21:09:32.631092 10.10.10.141:35512 -> 172.20.11.2:22
TCP TTL:64 TOS:0x0 ID:31113 IpLen:24 DgmLen:44
IP Options (1) => LSRR
******S* Seq: 0x68474C99  Ack: 0x0  Win: 0x200  TcpLen: 20

[**] [1:500:2] MISC source route lssr [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/18-21:09:37.740989 10.10.10.141:35512 -> 172.20.11.2:22
TCP TTL:64 TOS:0x0 ID:31113 IpLen:24 DgmLen:44
IP Options (1) => LSRR
******S* Seq: 0x68474C99  Ack: 0x0  Win: 0x200  TcpLen: 20
```

What rule triggered this alert? A quick grep in the rules directory showed the following entry in the file misc.rules (for the record, both EXTERNAL_NET and HOME_NET were set to *any*, meaning they match everything):

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC source route lssr";
    ipopts:lssr; reference:bugtraq,646; reference:cve,CVE-1999-0909;
    reference:arachnids,418; classtype:bad-unknown; sid:500; rev:2;)
```

12 Available at http://www.incidents.org/logs/Raw/README [5]

The rule triggers on all packets with the IP option LSRR, or “Loose Source Route and Record Route”. Consequently, such packets should exist in the binary log file. This is confirmed in figure 9. The tcpdump filter could be written to check specifically for LSRR (by adding “and ip[20] = 0x83”), but I wanted to see if there was any other traffic with IP options (it was not).

```
$ tcpdump -r 2003.12.15.7 -X -s 1514 -n -v "tcp and ip[0] & 0x0f > 5"

21:09:27.593273 IP (tos 0x0, ttl  64, id 31113, offset 0, flags [none], length:
44, optlength: 4 ( LSRR{#} EOL )) 10.10.10.141.35512 > 172.20.11.2.22: S [bad
tcp cksum a286 (->599d)!] 1749503129:1749503129(0) win 512

        0x0000:  4600 002c 7989 0000 4006 ad92 0a0a 0a8d  F..,y...@.......
        0x0010:  ac14 0b02 8303 0400 8ab8 0016 6847 4c99  ............hGL.
        0x0020:  0000 0000 5002 0200 a286 0000 0000       ....P.........
```

Fig 9 Packet with loose source route.

Two almost identical packets (the only differing fields are the IPID and ISN) follows at 21:09:32 and 21:09:37, which could indicate retries of some sort (normal TCP retries would probably not come every fifth second, though). A quick glance in the RFC[13] confirms that IP option 0x83 (131 decimal) is indeed LSRR.

Probability the source address was spoofed

There are three alternatives to take into consideration: Spoofed, not spoofed, or third party effect.

I am quite confident it is no third party effect. The Snort sensor is sitting on the same subnet as the attacking host, and it is sending out TCP packets with the SYN flag set; hence it is not a response coming back from somewhere.

Is the traffic spoofed by another host on the same subnet? While possible, after taking a look at the traffic directly preceding the LSRR (figure 10), I do not think that is the case, either.

13 RFC 791, p. 18. [6]

```
$ tcpdump -nc 3 -r 2003.12.15.7 \
    "host 10.10.10.141 and host 172.20.11.2 and tcp port 22"

21:08:43.859334 IP 10.10.10.141.32808 > 172.20.11.2.22: S 3285526535:3285526535
(0) win 5840 <mss 1460,sackOK,timestamp 45333 0,nop,wscale 0>

21:08:44.086622 IP 172.20.11.2.22 > 10.10.10.141.32808: S 4163270468:4163270468
(0) ack 3285526536 win 5792 <mss 1460,sackOK,timestamp 5038103 45333,nop,wscale
0>

21:08:44.086734 IP 10.10.10.141.32808 > 172.20.11.2.22: . ack 1 win 5840
<nop,nop,timestamp 45356 5038103>
```

Fig 10 Packet trace with TWH.

If the traffic had been spoofed, 10.10.10.141 would hardly have responded with a completing ACK; rather a RST would have been sent.

A quick note - it could be argued the LSRR traffic is spoofed, while the above is not. I am ruling out this alternative since it is the same MAC address, and the fact the above trace is from a TCP Connect scan, and the LSRR packets follows immediately after (there is approximately 50 seconds between them, and the scan was starting on port 0 and going upwards).

Description of attack

The attack is a reconnaissance scan produced by the vulnerability scanner Nessus[14]. The goal is to establish whether IP packets with the option LSRR (Loose Source Route and Record Route) will be allowed. The primary target is not really the remote host itself, but rather firewalls and routers on the way.

The documentation [8] for this specific Nessus plugin can be found at the following URL: http://cgi.nessus.org/plugins/dump.php3?id=11834

In my opinion, none of the Snort references[15] really applies to the Nessus plugin. More to the point is the CVE candidate - CAN-1999-0510 [9] – which simply states: "*A router or firewall allows source routed packets from arbitrary hosts*" (perhaps a bit broad, but I think it will serve if it ever is accepted as a CVE). The URL for this CAN is: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0510

Attack mechanism

I will begin to establish some of the problems associated with source routing (and loose source route, in particular). ISS[16] offers the following description [10]:

14 Nessus is open source and available at http://www.nessus.org [7]
15 More thourougly described in the correlations section.
16 Internet Security Systems.

LSRR can be used in a number of ways for hacking purposes. Sometimes machines will be on the Internet, but will not be reachable. (It may be using a private address like 10.0.0.1). However, there may be some other machine that is reachable to both sides that forwards packets. Someone can then reach that private machine from the Internet by source routing through that intermediate machine.

Furthermore, as the GCIA course material[17] states, source routing can be used to get spoofed packets back, as the return path usually will be the same one they arrived on (this is where the "Record Route" part of the name comes into play).

Here comes the reason I decided to take a closer look on the source routed traffic. Something odd with those packets caught my eye. According to the RFC, the first octet is the option, the second is the length (including the first three octets), and the third is the pointer (all specified hops are processed when the pointer is larger than the length). The IP header shown in figure 9 is 24 bytes long. The LSRR length value is 3, and the pointer 4. Fine. An IP address is four octets long in my book. So, how many addresses can be squeezed into the remaining length? Yup. None.

My next question was "Why would somebody want to do that?" It just did not look very dangerous. It was not even malformed (for example, the pointer byte could be manipulated in order to cause a denial of service in poorly written IP stacks).

Further review (manually, using Ethereal) on the captured data found the following packet sequence following the completion of a TWH (Three Way Handshake):

```
21:09:20.465151 IP 172.20.11.2.22 > 10.10.10.141.34048: P 1:24(23) ack 1 win
5792 <nop,nop,timestamp 5041733 48990>

        0x0000:  4500 004b b586 4000 3e06 bb79 ac14 0b02  E..K..@.>..y....
        0x0010:  0a0a 0a8d 0016 8500 fa21 5bc4 c5de 7fc8  .........![.....
        0x0020:  8018 16a0 58ea 0000 0101 080a 004c ee45  ....X........L.E
        0x0030:  0000 bf5e 5353 482d 312e 3939 2d4f 7065  ...^SSH-1.99-Ope
        0x0040:  6e53 5348 5f33 2e35 7031 0a              nSSH_3.5p1.

21:09:20.465272 IP 10.10.10.141.34048 > 172.20.11.2.22: . ack 24 win 5840
<nop,nop,timestamp 48994 5041733>

21:09:20.465539 IP 10.10.10.141.34048 > 172.20.11.2.22: P 1:23(22) ack 24 win
5840 <nop,nop,timestamp 48994 5041733>

        0x0000:  4500 004a c0a0 4000 4006 ae60 0a0a 0a8d  E..J..@.@..`....
        0x0010:  ac14 0b02 8500 0016 c5de 7fc8 fa21 5bdb  .............![.
        0x0020:  8018 16d0 935f 0000 0101 080a 0000 bf62  ....._.........b
        0x0030:  004c ee45 5353 482d 322e 302d 4e65 7373  .L.ESSH-2.0-Ness
        0x0040:  7573 5353 485f 312e 300a                 usSSH_1.0.
```

Fig 11 Three packets immediately following TWH (the second packet is shown without data).

17 Novak et al. section 8, p 25. [11]

Right! Two more bits of information. For one thing, it seems to be a real SSH server listening on the target, and for another, I got an idea what was generating the traffic: The well-known vulnerability scanner named Nessus!

At this point, it was logical to do a lab experiment with said vulnerability scanner. And, as a matter of fact, it did have a plugin for testing LSRR (located in the Firewalls plugin section, titled "Source routed packets").

It would seem that the attack not is directed at the target host itself, but rather at firewalls and other network devices in front of it. However, the purpose would still be to find means to compromise the target (or other hosts on the same network) by some sort of creative packet crafting.

As already demonstrated, there is a SSH service running at the target host, and it is not blocked by any filtering devices. If the LSRR packet arrived to it, a SYN/ACK packet would be returned (unless the host is configured to ignore such traffic).

A note though: the attack is not targeting port 22 for any other reason than it is known to be open from prior scanning. It is, in other words, not a SSH attack.

Correlations

While I have found no prior analysis on the Nessus LSRR scan, source routing by itself is certainly nothing new.

The Snort signature comes with three references:

1. http://www.whitehats.com/info/IDS418
   This reference [12] is not entirely to the point as it refers to UDP packets, while the trace described above is TCP. It does state that "*Source routing IP Options should be blocked by your firewall or routers"*, which is sound advice.

2. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0909
   This CVE [13] describes an attack against Microsoft Windows systems, where a source routed packet with an offset greater than the length will be passed on to the application layer even if source routing is disabled (as discussed previously, the offset pointer being greater than the length indicate that all routers in the list has been traversed).

3. http://www.securityfocus.com/bid/646
   This is the Security Focus description [14] of CVE-1999-0909. It is not very detailed, so I recommend the NAI advisory [15] instead: http://www.securityfocus.com/advisories/1761

As has already been discussed, I feel the CVE candidate CAN-1999-0510 is more relevant to this attack. The URL for this candidate is:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0510

Evidence of active targeting

This specific attack is directed at a host known to be up, and a service known to be responding. On the other hand, it seems to be just one of many different probes. But in any case, if someone starts a vulnerability scan directed at another internal host, it is probably not a wrong number. In my opinion, this is a case of active targeting.

Severity

I will discuss each parameter separately, and attempt to motivate the scores.
The formula that will be used is: Severity = (Criticality + Lethality) - (Host countermeasures + Network countermeasures).

Criticality

When I started to estimate the criticality, I found that I would like to know a bit more about the target host. In particular, did it offer any other services?

Figure 12 shows SYN-ACK packets from the target to the attacker.
Such packets indicate an open service[18].

```
$ tcpdump -n -r 2003.12.15.7 \
    "dst host 10.10.10.141 and src host 172.20.11.2 and tcp[13] = 0x12" \
    | awk '{print $3}' | sort | uniq -c

     19 172.20.11.2.22
     13 172.20.11.2.53
```

Fig 12 Responding services on IP 172.20.11.2.

From the result, I surmise the target is a DNS server, and that SSH is used to manage it remotely. Another assumption is that the target is a UNIX or Linux system, as it is running both SSH and DNS, but not any NetBIOS services.

Being a DNS server makes the host critical indeed. Even if it were "only" an internal DNS, a compromise could still severely affect the overall security (having control of a DNS server must be one of the most simple methods to do man-in-the-middle attacks). As such, I feel the criticality must be given the maximum score (5).

Lethality

As the attack described is a reconnaissance scan, and that I see no potential for a denial of service due to it, the lethality is very low (1). However, if the attack succeeded, it would give the attacker the information that other, more vicious attacks might be performed. Another thing I considered here is that source routed packets hardly is something new on the scene – it should not be a surprise to anybody.

18 TCP is described in RFC 793 [16].

Host and network countermeasures

Obviously, the attack is thwarted somewhere along its path. What I cannot tell is exactly where that does happen. As a result, I must do an educated guess and choose between three alternatives:

1. The host drops LSRR packets
2. A network device on the path drops the packets
3. Both the network and the host drops LSRR packets.

Remember, I know what version the SSH server is running: OpenSSH 3.5p1. Now, as stated in Part 1, that version has some problems associated with it[19]:

"September 16, 2003: OpenSSH Buffer Management bug"
“September 23, 2003: Portable OpenSSH Multiple PAM vulnerabilities”

If the administrator of the host does not keep the SSH daemon up to date, I will assume the worst and say that the host probably do allow LSRR packets (on the other hand, no TCP small services were running, so the box seems to be somewhat hardened).

If indeed it was a network device that blocked the traffic, it is also configured not to emit an ICMP error message. Such a message would give the attacker two valuable pieces of information: that the LSRR packet was dropped, and the IP address of the device (when testing in a lab, I got back an ICMP type 12, code 0, which is "parameter problem"[20]).

Thus, I will give the network counter-measure a score of 3 (even though it otherwise seems to be quite wide-open; for this attack it made it!) and the host countermeasure a 2 (even if the current administrator has lax patching routines, there are at least some indication that the host was sufficiently hardened when it was installed).

Result

The severity, then, would be (5 + 1) - (2 + 3), which equals 1.

The outcome is very dependent on the network countermeasures. If I were to grade the *overall* network protections (i.e. not pertaining to just this attack), it would probably be 1 or 2 instead, which naturally would effect the severity.

Defensive recommendation

19 Source: the OpenSSH security page, located at http://www.openssh.org/security.html [17]
20 One might expect a “Destination Unreachable: Source Route Failed” message (type 3, code 5). However, the parameter problem pointer was 40, which in this case was the first octet of the IP options. The router might have been configured to disallow all IP options rather than just LSRR.

As most firewall analysts agree[21], source routing should under no circumstances be allowed into the network. Apparently, the attack was unsuccessful since the attacker failed to get a response back. But as stated in the Severity section, it is not clear at which point the attack was stopped.

The first defensive recommendation would therefore be to check the target host and all network devices on the path to verify they block source routed packets. All such packets should be discarded silently, as to not allow potential attackers to receive any extra information about the network layout.

The second one is actually more important, but it does not relate exactly to the attack. The target host is running an old SSH daemon with known vulnerabilities. It should be upgraded to the latest one as soon as possible.

I can of course only speculate on the internal network design and local policies, but it might also be a good idea to restrict SSH access for all hosts except those located on management networks, especially when it comes to such important systems as DNS servers. It is of course a possibility that 10.10.10.0/24 *is* a management network, and 10.10.10.141 is an administrator's box testing the target system for vulnerabilities.

Multiple choice question

Which of the following is *not* true in regard to loose source routing?

A) The RFC terminology is LSRR - Loose Source Route and Record Route.
B) It is considered a “bad thing”, even though there are legitimate uses for it.
C) It can used by an attacker to get responses from spoofed packets.
D) When set, the total length of the IP options field must be four bytes.

Correct answer: D

(As has been discussed, a length of three bytes means an empty list; four bytes would be malformed as it could contain only the first octet of an IP address).

Response from the incidents.org mailinglist

I submitted my analysis to the incidents list at April 22. The URL is listed below:
http://www.dshield.org/pipermail/intrusions/2004-April/007902.php [19]

At April 28, I resubmitted the analysis after having done some corrections:
http://www.dshield.org/pipermail/intrusions/2004-April/007945.php [20]

Unfortunately, at the time of submission, no respone has been received on any of the two postings, so I am unable to include this or any subsequent defense.

21 The earliest reference I have found is the "Firewalls and Internet Security" book by Cheswick and Bellovin (p 114), which was published in 1994 [18].

2.2. Source of trace for detects 2 and 3

Detects two and three were captured at an undisclosed site. Below (figure 13) is a sanitised network diagram showing involved hosts. This section effectively replaces "Source of trace" for the remaining two detects. Detects are recorded by the Snort sensor which is running Snort [0] version 2.1.0 on a GNU/Linux system.

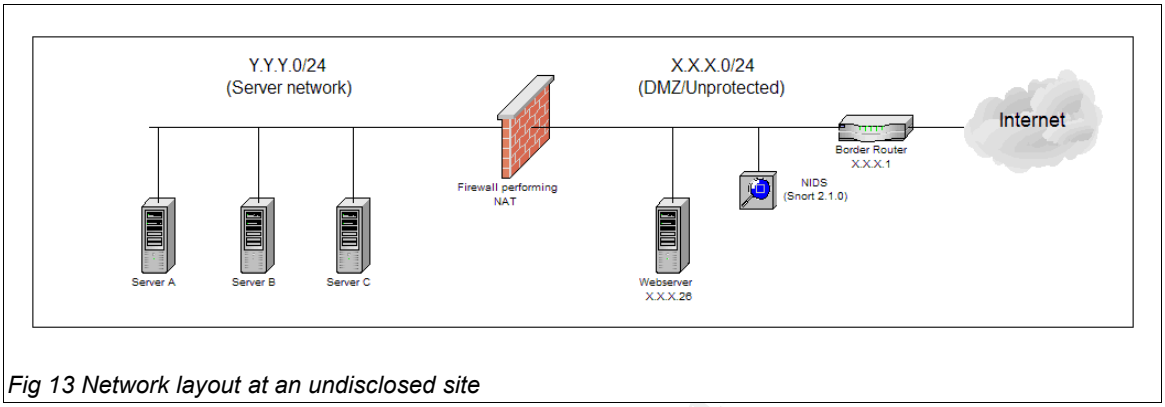


Fig 13 Network layout at an undisclosed site

Please be aware that IP addresses has been obfuscated where appropriate to protect the company in question.

2.3. Detect 2 – Microsoft WKSSVC.DLL Buffer Overflow Attack

Detect was generated by

One of the alerts looked thus:

```
[**] [1:653:6] SHELLCODE x86 unicode NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
04/05-17:59:18.713828 195.29.53.74:4035 -> X.X.X.26:139
TCP TTL:118 TOS:0x0 ID:57532 IpLen:20 DgmLen:1468 DF
***AP*** Seq: 0xCBE163D  Ack: 0x8E8F8783  Win: 0x2017  TcpLen: 20
```

It was generated by the following rule (from the shellcode.rules file):

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any
            (msg:"SHELLCODE x86 unicode NOOP";
            content: "|9000900090009000|";
            classtype:shellcode-detect; sid:653; rev:6;)
```

The rule looks at the packet payload for the sequence of 90009000 (and so on), which is an UTF-16 LE[22] encoded NOP[23] for the x86 processor family. This rule generated several false positives from HTTP sessions; however this particular

22 UTF-16 is part of Unicode. LE stands for Little Endian; UTF-16 and UTF-32 can be encoded both as Big and Little Endian. Microsoft Windows generally use Little Endian. [21]

23 NOP (and also NOOP) stands for "No Operation" and is used for telling the CPU that no particular operation is desired for the current instruction. Several NOPs in a sequence is called a NOP Sled, and is used to broaden the valid return range when writing buffer overflows.

packet (figure 13) is directed towards the NetBIOS Session Service, which immediately makes it more interesting. Worth to note is that the rule is wont to trigger on Big Endian NOPs as well, due to the fact that the NOP sequence in most cases will be longer than the one in the rule (thus, the sequence could be 00 90 00 90 and so on, and still trigger, as long as there is at least two more NOPs than specified in the rule). The packet it triggered on was Little Endian and contained 42 NOPs in the first sequence. This can be seen in figure 14.

```
17:59:18.713828 IP 195.29.53.74.4035 > X.X.X.26.139: P
213784125:213785553(1428) ack 2391771011 win 8215 NBT Packet

        0x0000:  4500 05bc e0bc 4000 7606 XXXX c31d 354a  E.....@.v.)...5J
        0x0010:  XXXX XXXX 0fc3 008b 0cbe 163d 8e8f 8783  .B9........=....
        0x0020:  5018 2017 XXXX 0000 4100 4100 4100 4100  P.......A.A.A.A.
        0x0030:  4100 4100 4100 4100 4100 4100 4100 4100  A.A.A.A.A.A.A.A.
..
..
        0x04e0:  4100 4100 4100 4100 4100 4100 4100 4100  A.A.A.A.A.A.A.A.
        0x04f0:  4100 4100 4100 4100 4100 4100 9000 9000  A.A.A.A.A.A.....
        0x0500:  9000 9000 9000 9000 9000 9000 9000 9000  ................
        0x0510:  9000 9000 9000 9000 9000 9000 9000 9000  ................
        0x0520:  9000 9000 9000 9000 9000 9000 9000 9000  ................
        0x0530:  9000 9000 9000 9000 9000 9000 9000 9000  ................
        0x0540:  9000 9000 9000 9000 9000 9000 9000 9000  ................
        0x0550:  9000 9000 4700 a700 1800 7500 9000 9000  ....G.....u.....
        0x0560:  9000 9000 9000 9000 9000 9000 9000 9000  ................
        0x0570:  9000 9000 9000 9000 9000 9000 eb00 1900  ................
        0x0580:  5e00 3100 c900 8100 e900 3020 d902 d902  ^.1.......0.....
        0x0590:  d902 8100 3600 ac20 7c01 3200 1d20 8100  ....6...|.2.....
        0x05a0:  ee00 fc00 d902 d902 d902 e200 4801 eb00  ............H...
        0x05b0:  0500 0d01 e200 d902 d902 d902            ............
```

Fig 14 Possible executable code?

Probability the source address was spoofed

This analysis is quite difficult since I only have traces of packets going from the attacker to the target. Simply put, I have no hard proof that the target responded. Nevertheless, there is an indication that the triggering packet was part of a TCP session (it has the same source port as an incoming SYN, and the portscan log showed that the source port increased after each new session, thus conforming with the TCP standard). There are also further indications that the attacking host got responses back, and subsequently made use of that information. My analysis, therefore, is that it is unlikely that the source address is spoofed.

Description of attack

I think the attack is targeting the WKSSVC.DLL (Workstation Service) on Microsoft Windows 2000 and XP systems. The attack can be triggered by sending a long unicode string to port 139 or 445 (both TCP). If succeeded, a buffer overflow will occur in a vsprintf() call as no prior bound checking is performed.

The vulnerability was discovered by eEye Digital Security and published in November 12 last year (2003). The URL to the advisory is listed below:
http://www.eeye.com/html/Research/Advisories/AD20031111.html [22]

Microsoft released a patch for the problem. Additional information is available in Microsoft's Security Bulletin MS03-49:
http://www.microsoft.com/technet/security/bulletin/MS03-049.mspx [23]

The CVE candidate for this vulnerability is CAN-2003-0812 [24].

Attack mechanism

The first thing to decide: is it a false positive or not? To do this, we need to look at the entire packet (please refer back to figure 14).

There is a whole lot of 'A's in that packet. And each 'A' is followed by 0, which usually is used for NULL termination of strings. However, the pattern will also be seen when the ASCII 'A' is encoded in UTF-16 LE (which happens to be used by Microsoft Windows systems). To be exact, there are 618 occurences of the "41 00" pattern, meaning 618 'A's when converted to ASCII (according to eEye, the ASCII conversion will happen before the overflow occurs).

After all those "41 00"s, there comes a bunch of x86 unicode NOP's (an UTF-16 LE encoded NOP is “90 00”, which differs from the ASCII one that is “90”. Then some other data appears, followed by more NOPs, and even more data. This looks like an attempted buffer overflow to me.

The analysis is made more complicated by the fact this is the only packet in the sequence. Did any other preceed it? What came after?

Luckily, the attacking host did some other things as well. In fact, it did so much other stuff that Snort's portscan preprocessor kicked in, and thus also logged the existence of related packets (even though it did not log the entire packets; at this point I was grateful for anything). The related packets are shown below:

```
Apr  5 17:59:05 195.29.53.74:4022 -> X.X.X.26:139 SYN ******S*
Apr  5 17:59:08 195.29.53.74:4035 -> X.X.X.26:139 SYN ******S*
Apr  5 17:59:11 195.29.53.74:4035 -> X.X.X.26:139 SYN ******S*
Apr  5 17:59:53 195.29.53.74:4230 -> X.X.X.26:139 SYN ******S*
Apr  5 17:59:56 195.29.53.74:4230 -> X.X.X.26:139 SYN ******S*
Apr  5 17:59:59 195.29.53.74:4248 -> X.X.X.26:139 SYN ******S*
Apr  5 18:00:04 195.29.53.74:4270 -> X.X.X.26:139 SYN ******S*
Apr  5 18:00:49 195.29.53.74:4404 -> X.X.X.26:139 SYN ******S*
Apr  5 18:00:59 195.29.53.74:4450 -> X.X.X.26:139 SYN ******S*
Apr  5 18:01:00 195.29.53.74:4452 -> X.X.X.26:139 SYN ******S*
```

Fig 15 Excerpt from portscan.log

With the portscan log at hand, I can do a speculation on what occured here.

1. It all begins with a scan (either SYN or a full connect – I cannot tell).
2. The target responds - the attacker knows the port is open.
3. The attack starts. A SYN is sent. For some reason, no respone is forthcoming.
4. Another SYN with same source port. Looks like a normal TCP retry (same source port and three seconds after the first packet). TCP establishment occurs. Perhaps some packets after the TWH, but I doubt it.
5. The attack takes place. It may or may not have succeeded. The TCP session may or may not have been terminated gracefully.
6. After the attack, a new session is started. The response never returned so a TCP retry is sent. Then establishment. From looking at the source ports, the attacking host appears to be doing lots of things at the same time. This may explain the retries – the attacker's host could be too busy to process all answers.
7. The last thing for destionation port 139 is five additional sessions. I do not know about the purpose of those. Checking for success, perhaps?

I would give a kingdom for a complete packet trace between those hosts. The problem is, as I have no kingdom to spare at the moment, I will have to resort to some additional research instead.

For example, were any other ports targeted?

```
$ grep 195.29.53.74 portscan.log | grep X.X.X.26 | awk '{print $6}'
    | sort | uniq -c

      3 X.X.X.26:135
     10 X.X.X.26:139
      1 X.X.X.26:26
      1 X.X.X.26:4444
      8 X.X.X.26:445
      2 X.X.X.26:80
```

Fig 16 Other scanned ports from source 195.29.53.74

As can be seen, a few other ports were scanned. Were any of them attacked?

```
$ grep -E '195\.29\.53\.74:[[:digit:]]+ -> X\.X\.X\.26:[[:digit:]]+' \
    alert -B 2 | sed -e 's/^--$//g'

[**] [1:2192:1] NETBIOS DCERPC ISystemActivator bind attempt [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
04/05-17:58:48.222574 195.29.53.74:3920 -> X.X.X.26:135
```

Fig 17 Excerpt of other attacks from source 195.29.53.74 (please note that GNU grep is required)

At this point, some information regarding the server was aquired. It was a NT 4 system (which did not come as a surprise considering the targeted ports), running the IIS web server for a few domains.

Of the targeted ports, only 135 and 139 were open (verified with netstat on the host itself). The reason port 80 did not respond is that the web servers ran as virtual hosts bound to different IP addresses.[24]

The buffer overflow was targeted only towards port 139. This means we do not have to consider vulnerabilites that can be targeted at both 135 and 139. We still need to consider the other ports though – especially port 445 as this is also used for authentication, but in W2K domains. Another important tidbit of data to keep in mind, port 135 was both scanned and attacked before port 139, which adds to the conclusion that port 139 was deliberately targeted for this particular attack.

With this information, it was possible to start to search for corresponding vulnerabilites and attacks. As for vulnerability, the one discussed in the Attack Description seemed to fit nicely (it targets both ports 139 and 445, and the attack is carried out by a long Unicode string that is later on converted to ASCII).

If that is correct, then the likelihood of success is small, even if the system would not be up to recommended patch levels (according to both eEye and Microsoft, NT4 is not vulnerable).

What becomes interesting next is the status of the server. Are all recommended security patches applied, and was it compromised? If it were, my analysis would be wrong.

Well, nothing untoward seems to be running on the system, even though there are a few unpatched vulnerabilites on it (SP6a is not applied, for one thing).

There are several reports of this attack existing in the wild, Unfortunately, the only way to be sure of my guess would be to test the attack on a W2K or XP system and see what happens, or to decode the UTF data and try to match the 'A's and shellcode with public exploits. Both alternatives have their drawbacks, and I feel it is outside the scope for this detect.

Correlations

The reference section of the CVE candidate contains listings to several advisories (which is some cases started discussions on the mailing lists where they were publicised). The URL for the candidate is:

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0812 [24]

I have not found any other reports on this exact pattern.

The attacking address, which is 195.29.53.74, resolves to ad18-m74.net.htnet.hr. The netblock belongs to Croatian Telecom, which fits the .hr ccTLD perfectly. I have found no evidence that this system has participated in other attacks.

24 These were also attacked, but that does not belong to this analysis.

Evidence of active targeting

To sum up my findings in the Attack Mechanism section: The attack is targeted at port 139 (and probably 445 as well) when known to be open. The attack was preceeded by a portscan to find open ports. So yes, the attack is somewhat targeted, but not as scary as it would have been if it came without the portscan.

Severity

Criticality

The system is a public web server, and contains no confidential information. Less than 10 domains are running on it. A compromise could cause bad will if the web pages were defaced or became unavailable for a longer period of time. Score: 2

Lethality

A buffer overflow targeted at the OS. It could yield SYSTEM[25] level access if succeeded. I am reserving myself a bit though: I have not verified that the shellcode works, but if I have connected the attack to the correct vulnerability, the potential impact is arbitrary commands executed as SYSTEM. Score 5.

Network countermeasures

There is no firewall in front of the system. The router does no ingress filtering at all. The score here must unfortunately be the lowest possible. Score: 1

Host countermeasures

The system is not up to date on patches. No internal firewall is protecting it and the targeted port is listening. The reason I give a 2 here (instead of 1) is that the system is too old for the attack – it does not work against NT 4. That is not a whole lot of defenses, though. Score: 2

Result

Severity: (2 + 5) - (1 + 2) = 4. This is kind of interesting given the low criticality. On the other hand, it is only fair. The only thing saving this box from compromise is the fact the attack targeted W2K and WXP boxes rather than NT4. And a rooted box is never good news. Once on the same network segment as the firewall, attacks such as ARP spoofing becomes possible.

Defensive recommendation

I will list some of the problems associated with the system and the network, and the suggested corrective actions to resolve each issue.

25 SYSTEM level access is the Windows equivalent to root in UNIX systems.

A Microsoft IIS server is located outside of the corporate firewall

I am not attempting to do any Microsoft bashing here, but it is well-known that both the OS (NT4) and the IIS web server has suffered from several critical vulnerabilities. New ones are likely to be discovered.

Common sense says such systems should be placed behind a firewall (in fact, all systems should be behind a firewall, no matter how seldom vulnerabilites are discovered).

It is not part of a domain, but it is still listening on NetBIOS ports

The recommendation is to turn of NetBIOS altogether in order to avoid future attacks or information leakage. A service that does not run is hard to attack.

NetBIOS is not blocked at the border router

Well, it should be, really. There is few reasons to allow it, and even if the need existed, it would be prudent to allow access only through a VPN.

Old software versions

The server is running IIS version 4, on NT4. Both are old, and NT4 is approaching its End of Life statement (it has in fact been extended several times due to many corporations lack of interest to upgrade). The organisation should consider how to upgrade the system before the software expires (the biggest security implication of expired software would be no more security patches).

System not up-to-date on patches

First, all recommended security patches should be applied. This will fix the symptom. As for the real problem, a patch policy should be written if not already existing. If it exists, it should either be revised or be more stringently followed.

Multiple choice question

When used in conjunction with a buffer overflow, what purpose fits best for the following data sequence: “90 00 90 00 90 00 90 00 90 00”?

A) The contents of the buffer to be overflowed is oftenly filled with it.
B) It is a common NOP code for the UltraSparc II ISA.
C) It is a common NOP code for most x86 ISAs
D) It is an x86 ISA NOP code encoded with UTF-16 (Unicode)

Correct answer: D

2.4. Detect 3 – The Deadhat.B Worm

Detect was generated by

The following detects where found in the Snort alert file. Well, in fact, several more of these were found, these are only the first four in a sequence of about 85.

```
[**] [1:615:5] SCAN SOCKS Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
03/31-11:55:18.474173 218.5.20.107:22002 -> X.X.X.2:1080
TCP TTL:108 TOS:0x0 ID:64668 IpLen:20 DgmLen:40
******S* Seq: 0x4812  Ack: 0x6F7D  Win: 0x7A67  TcpLen: 20
[Xref => http://help.undernet.org/proxyscan/]

[**] [1:618:5] SCAN Squid Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
03/31-11:55:18.674354 218.5.20.107:22002 -> X.X.X.2:3128
TCP TTL:109 TOS:0x0 ID:64680 IpLen:20 DgmLen:40
******S* Seq: 0x3789  Ack: 0x69C0  Win: 0x7EC9  TcpLen: 20

[**] [1:615:5] SCAN SOCKS Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
03/31-11:55:18.874769 218.5.20.107:22002 -> X.X.X.3:1080
TCP TTL:122 TOS:0x0 ID:64690 IpLen:20 DgmLen:40
******S* Seq: 0x2700  Ack: 0x6403  Win: 0x32B  TcpLen: 20
[Xref => http://help.undernet.org/proxyscan/]

[**] [1:618:5] SCAN Squid Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
03/31-11:55:19.074799 218.5.20.107:22002 -> X.X.X.3:3128
TCP TTL:123 TOS:0x0 ID:64700 IpLen:20 DgmLen:40
******S* Seq: 0x7476  Ack: 0x691  Win: 0x4D56  TcpLen: 20
```

The following rules are included in the scan.rules file, which is part of the installation:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 3128 (msg:"SCAN Squid Proxy attempt";
    stateless; flags:S,12; classtype:attempted-recon; sid:618; rev:5;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 1080 (msg:"SCAN SOCKS Proxy attempt";
    stateless; flags:S,12; reference:url,help.undernet.org/proxyscan/;
    classtype:attempted-recon; sid:615; rev:5;)
```

The rules checks for inbound TCP packets with the SYN flag set (ignoring ECN bits), and does not care about prior state of the session (this is done with the *stateless* keyword).The packets the rules are triggering on is showed in figure 18.

```
11:55:18.474173 IP 218.5.20.107.22002 > X.X.X.2.1080: S
11:55:18.674354 IP 218.5.20.107.22002 > X.X.X.2.3128: S
11:55:18.874769 IP 218.5.20.107.22002 > X.X.X.3.1080: S
11:55:19.074799 IP 218.5.20.107.22002 > X.X.X.3.3128: S
```

Fig 18 Packets triggering the Snort rules.

Probability the source address was spoofed

There is a possibility that the source address is spoofed; there are evidence of other IP header fields being crafted. On the other hand, the analysis shows that this is a worm attempting to spread, and it is using TCP to communicate. The network traces confirm that two-part communication is taking place. As such, my conclusion must be that this traffic not is being spoofed. This conclusion was later verified when running the worm in a contained test environment.

Description of attack

The attack is a worm targeting backdoors left in place by various MyDoom malwares[26]. Symantec's name is W32.HLLW.Deadhat.B [25]. It is reputed to have several attack vectors, but my main concern is its method of portscanning and then uploading a file for execution. When executed by the MyDoom backdor, it inactivates the MyDoom binary, and adds itself to the registry to get automatically started after a system reboot. As its name indicate, it targets Win32 systems.

It does have some characteristics that provides for easy identification:

- It scans TCP ports 1080, 3127, 3128, 10080.
- Source port is usually fixed to 22002.
- It establishes a new backdoor on port 2766.
- The TTLs are varying wildly between packets.

The worm does not cause any immediate damage (such as destroying data). However, the scanning seem to be quite network intense, and might cause denial of service conditions, especially at sites with limited bandwidth.

Attack mechanism

The one thing that really caught my interest was the source port, which was constant, and not increasing, as it should be for most normal applications. My initial thought was that I was seeing some sort of proxy scanning generated by a tool like nmap[27] which allows for user modified source ports.

It was at this stage I thought this might be worth to write about, and I took a look at the Intrusions mailinglist to see if it had already been covered. I found some other reports (these are covered in the correlations section), but no in-depth analysis. One of the posts there gave me an additional clue. It was not only the ports 3128 and 1080 that was being targeted, but also 3127 and 10080. I got access to the firewall logs, and managed to verify that was indeed the case.

26 It is getting harder and harder to say if something is a virus or a worm since they use techniques from both categories. Thus the term malware is becoming more and more favoured. For my part, I still think a worm is something that propagates through the network without user intervention (such as clicking on attachments or viewing it in the preview pane), so that will be my meaning whenever I use the term 'worm'.

27 Available from http://www.insecure.org/nmap [26]

Excerpt from Firewall Log (the log does not show TTLs)							
Date	***Time***	***Action***	***Proto***	***SRC IP***	***DST IP***	***DPORT***	***SPORT***
6Apr2004	16:29:39	drop	tcp	62.201.64.66	10.1.2.2	3127	22002
6Apr2004	16:29:39	drop	tcp	62.201.64.66	10.1.2.2	1080	22002
6Apr2004	16:29:39	drop	tcp	62.201.64.66	10.1.2.2	10080	22002
6Apr2004	16:29:39	drop	tcp	62.201.64.66	10.1.2.2	3128	22002

A new snort rule was added to capture the complete scan. The complete scan pattern is shown in figure 19. The TTL values are really interesting.

```
07:34:12.514505 IP (ttl 114, length: 40) 192.168.1.5.22002 > X.X.X.2.3127: S
07:34:12.568048 IP (ttl 111, length: 40) 192.168.1.5.22002 > X.X.X.2.1080: S
07:34:12.645453 IP (ttl 127, length: 40) 192.168.1.5.22002 > X.X.X.2.10080: S
07:34:12.712233 IP (ttl 124, length: 40) 192.168.1.5.22002 > X.X.X.2.3128: S

07:34:12.785866 IP (ttl 109, length: 40) 192.168.1.5.22002 > X.X.X.3.3127: S
07:34:13.005955 IP (ttl 125, length: 40) 192.168.1.5.22002 > X.X.X.3.1080: S
07:34:13.098609 IP (ttl 114, length: 40) 192.168.1.5.22002 > X.X.X.3.10080: S
07:34:13.232869 IP (ttl 119, length: 40) 192.168.1.5.22002 > X.X.X.3.3128: S
```

Fig 19 Scanning pattern – notice the TTLs.

I do expect the TTL to vary, but not that much. It would indicate an incredibly unstable connection. And when I see the same pattern from three different locations (not showed in the trace), I go “hmm!”. The word “crafted” comes to mind. But why? Since someone obviously thought it worth the effort, there is probably some reason behind it. My guess is that it is supposed to stop us analysts from pinpointing the location, or distance. It might be wrong, but it is the only thing I can come up with right now.

I got curious, and let a host listen on port 3127 over a night, and captured all data with a sniffer. The scan hit, my host dutifully answered SYN-ACK, and got a RST back. Immediately after, another connection was made from the same source, but now with a normal source port (2241). Figure 20 shows the beginning of the conversion (the TWH has already taken place).

```
01:02:49.827280 IP 67.97.205.104.2241 > X.X.X.X.3127: P 1:2(1) ack 1

0x0000   4500 0029 ba90 4000 7306 XXXX 4361 cd68        E..)..@.s.@*Ca.h
0x0010   XXXX XXXX 08c1 0c37 79e3 c5e2 581c 190b        .B9....7y...X...
0x0020   5018 faf0 XXXX 0000 8500 0000 0000             P.............

01:02:49.827374 IP X.X.X.X.3127 > 67.97.205.104.2241: . ack 2 win

01:02:49.840318 IP 67.97.205.104.2241 > X.X.X.X.3127: . 2:1462(1460) ack 1

0x0000   4500 05dc ba91 4000 7306 XXXX 4361 cd68        E.....@.s.:vCa.h
0x0010   XXXX XXXX 08c1 0c37 79e3 c5e3 581c 190b        .B9....7y...X...
0x0020   5010 faf0 XXXX 0000 133c 9ea2 4d5a 9000        P........<..MZ..
0x0030   0300 0000 0400 0000 ffff 0000 b800 0000        ................
0x0040   0000 0000 4000 0000 0000 0000 0000 0000        ....@...........
0x0050   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0060   0000 0000 0000 0000 f000 0000 0e1f ba0e        ................
0x0070   00b4 09cd 21b8 014c cd21 5468 6973 2070        ....!..L.!This.p
0x0080   726f 6772 616d 2063 616e 6e6f 7420 6265        rogram.cannot.be
0x0090   2072 756e 2069 6e20 444f 5320 6d6f 6465        .run.in.DOS.mode
```

Fig 20 Data exchange after a scanned port proved to be open.

As can be seen, first thing after the TWH, there comes a packet (PSH) with only one byte: 0x85. In the next packet, the following bytes comes: 0x13, 0x3C, 0x9E and 0xA2. I am not sure what these are for, probably some kind of instruction or length. After those, however, things begin to clear up. The next two bytes is 0x4D and 0x5A. Or, as it is in ASCII: MZ (anyone hear bells ringing?) Oh, and the string "*This program cannot be run in DOS mode"* is there too.

I was able to extract a fully functional PE[28] from the tcpdump logfile. The file size was 56 832 bytes, which is consistent with Symantec's report.

The next thing that is supposed to happen is that the uploaded binary is to be executed by a backdoor already in place (netcat on a non-windows system was not obliged to comply with the wishes of the worm).

However, if the file were to be executed on the Windows system, it would do several things, and few of them good (I set up an isolated network with a fresh Windows XP system to act as my sacrifice box):

Commence Scanning

Immediately after execution, it began the scanning. A certain pattern can be seen:

- It randomizes five sets of two bytes (denoted *X* and Y) used for the first two fields of the IP addresses. Then it begins to scan X.Y.0.0 on all four ports, continues to X.Y.0.1, and so on, for all five sets simultaneously.

- When X.Y.0.255 is reached, it goes over to X.Y.1.0 and starts over. In other words, each set contains 65,536 addresses. It spews out approximately one packet every 0.02 second.

28 Portable Executable

- I do not know what happens when it reaches X.Y.255.255. I assume one of:
  - New randomize (this is where I would place my bet).
  - Fall silent.
  - Increase the second octet (and ultimately the first one).

- The initial analysis that the TTLs were forged was quite correct. In the lab, i.e. on the originating subnet, I saw no TTL greater than 141, and no lesser than 122. This is also where I verified it did not spoof the source addresses.

Shutdown MyDoom processes

A copy of MyDoom-A [27] was retrieved from a nearby virus quarantine, and was executed on the system (after the date was set to January[29]). A listening port on address tcp/3127 appeared.

Then the new executable was run again. Port 3127 disappeared. However, the files installed by MyDoom was left intact, and so was the registry entries). It only stopped the backdoor process.

My findings disagree with Symantec's description, as the MyDoom registry entries or files not were removed. However, the MyDoom-A process will be started before Deadhat.B after reboot, which means it will be shut down again.

No other versions of MyDoom were tested, but I do not doubt they would be stopped as well (why else scan for the other three ports?)

Ensure survival through system reboots

It copied itself to C:\WINDOWS\SYSTEM32\MSGSRV32.EXE and created a registry entry in the Run folder to reload after reboot.

Establish new backdoor

It closes one hole, but opens another. A new listening port is opened at tcp/2766. When connected to with telnet or netcat, it responds with:
“SSH-2.0-OpenSSH_3.7.1p2 \m/”

It looks almost correct. Except the fact that normal OpenSSH servers do not terminate with “\m/”.

And it is not SSH either, of course. When trying to connect with a SSH client, I get a "connection closed" message back.

Final observations

An odd thing: There were also scans going to the same target ports, but without TTL or source port crafting. Those packets were very few in comparision, but did still appear. Why? I have no idea. The TTL on these were the OS default of 128.

29 MyDoom-A will not spread after February 12 [27]

I am not very good at debugging executables myself (which is why it is very neat to have friends who are). It appears the running process has capability to detect if someone is debugging it. Result: the logged in XP user was suddenly logged off! Harmless, but still quite fun, if one indeed is allowed to say that about malware. This is probably due to the compression/encryption rather than the worm itself. According to Panda[30], tElock v0.98 was used for this purpose.

Correlations

The scanning pattern has been reported in various postings from Ken Connelly, of which the below post is the first one (dated February 12):

http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00105.html [29]

The first [public] query on what this traffic actually was came from Michael Schwartzkopff in February 20 (although only ports 1080 and 3128 were reported):

http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00126.html [30]

Finally, Carl Gibbons reports the pattern as well (March 8):

http://cert.uni-stuttgart.de/archive/intrusions/2004/03/msg00032.html [31]

As the scanning is made by a worm, I felt it would not add anything to the analysis by attempting to correlate source addresses. It suffice to say that most of the scans I have seen comes from various ISP:s, distributed on several countries.

Symantec, in addition to most other antivirus vendors, has a description of this worm. Unfortunately, their analysis does not mention the portscanning behaviour more than to state "*Attempts to connect to sequential IP addresses on TCP ports 3127, 3128, and 1080 (ports that the Mydoom worm used)*". I see no mention of port 10080 (I am confident that it is the same binary, as it does have the same file size and otherwise matches the described behaviour).
Even so, Symantec has the most accurate description I have found.

http://securityresponse.symantec.com/avcenter/venc/data/ (line wrapped)
w32.hllw.deadhat.b.html [25]

I did post an analysis on this worm on the intrusions list, in hope it could be useful to somebody before the completion of this practical. This posting is available in DShield's archive, URL listed below:

http://www.dshield.org/pipermail/intrusions/2004-April/007888.php [32]

Evidence of active targeting

Normally, worms do not perform active targeting (unless, as has been seen in some cases lately, they try to perform a DoS against a third party). As has been showed,

30 http://www.pandasoftware.com/virus_info/encyclopedia/overview.aspx?lst=det&idvirus=44590 [28]

the target selection is entirely random. My conclusion is that there is no active targeting associated with this attack.

Severity

Criticality

The scan hit several important systems on the network, among them a DNS server, firewall, and a VPN endpoint. This section is therefore awarded full score (5).

Lethality

If the attack were successful, it would cause remote system access to everyone with access to the listening port. This section is also awarded 5 points.

Network countermeasures

All critical hosts were placed behind a firewall. None of the target ports were allowed through it. The design of the rulebase was “drop what is not explicitly allowed”. The score for network countermeasures is set to 4.

Host countermeasures

These varied between the hosts. Many of them were UNIX systems, making it virtually impossible for them to execute a MSDOS PE file. Most of them had recent operating systems installed. Among the Microsoft Windows systems, some where found to have antivirus products installed. None were infected with MyDoom (of any version), making this specific attack very hard to succeed.

With the above in mind, I have given this section 3 points (due to some inconsistencies with the regard to existing patch policies).

Result

The severity of this attack is 3 ((5 + 5) – (4 + 3)).

Defensive recommendation

Some of the source addresses comes from the reserved RFC 1918 [33] address space. That tells us two things. There is not enough egress filtering being done near the source. This is outside of our control; however, as these packets also arrive unmolested, we can also also assume there is a lack of ingress filtering on our part. This in turn could mean there is insufficient egress filtering as well. The recommendation, then, is to ensure that both egress and ingress filtering is performed (at the very least, anti-spoofing measures should be employed – there is no reason to allow RFC 1918 addresses coming into the network, nor should any source addresses not part of the network be let out onto the Internet).

Another thing would be to create new Snort rules, and configure them to watch for internal system responding on ports 1080, 3127, 3128, 10080 and 2766 (Any Squid proxies would have to be exempt for the ports they actually use).

It might also be prudent to watch for such traffic going out. This could indicate an internal compromise (by way of IRC, P2P, or other means[31]), or simply an internal user scanning external systems (which is a bad thing in almost all cases).

Finally, as this worm targets backdoors left in place by email borne viruses, it would be in an organisation's best interest to ensure all Microsoft Windows systems have up-to-date antivirus definitions, and that its employees are receiving on-going security education. A policy concerning email attachments should also exist.

Multiple choice question

TTL can be used in many ways. Which of the following statements is true?

A) TTL is defined in both the TCP and UDP headers
B) The TTL values will always be identical for all packets within a TCP session.
C) 128 is a common value for packets leaving a host (i.e. the initial value)
D) A common misconception is that TTLs is used by the Windows tracert command.

Correct answer: C

(I am attempting to trick the unwary to answer “D”. Windows tracert does differ, truly enough, but only because it is using ICMP instead of UDP high-ports. TTL is manipulated in the same way on both platforms. Many UNIX traceroute commands has an option to use ICMP as well)

31 This is covered in more detail by the antivirus vendors.

3. Part 3 – Analysis on .edu data

3.1. Executive Summary

This report is the result of an analysis performed on data produced by the on-site network intrusion detection system (NIDS). The data spanned a period of five days.

The following events were processed:

Type	*Description*	*Count*
Scans	Packets parts of reconnaissance probes	15,684,914
Alerts	Packets marked as possible attacks	93,538
OOS	"Out of Spec" - suspicious packets, but not necessarily attacks	5,896

The amount of recorded attacks and scans are large for a five day period. However, quite a few of them boils down to false alarms. This does not mean that all of them were so; in fact, there is evidence of internal systems that are infected by worms. Others have been found to perform network activity of questionable nature. More information about these systems, along with corrective recommendations, can be found in the "Defensive Recommendations" section.

It should be pointed out that far from every event was analysed in depth. The data set was simply too large. The methodology instead focuses on the Ten Top Talkers of each event type (scans, alerts and oos), but other systems and events of relevance have been covered where applicable.

The report begins with the defensive recommendations, and is followed by a link graph which reflects some of the activities that took place during the data set. Then comes a list of the files that were analysed, and the Top Ten Talkers for each event type. Next is the network registry information about five external systems that stood out during the assessment. After this comes four sections of more technical nature, and where more in-depth knowledge of network protocols and intrusion detection may be useful: activity which originates from the internal network, activity from external hosts, analysis on scanning activity, and a summary of abnormal traffic (“out of spec”). The report ends with a description on how the analysis was performed.

It should be noted that all fully qualified host names reflect the information at the time of writing. As the names are easy to change, and outside of control, they are only an indication of the remote identity. A malicious individual or organisation with control of the reverse DNS zone of their network may easily change any of their IP-addresses to resolve to "www.sans.org" or any other arbitrary name (note that it will not make www.sans.org to resolve to the IP - it is the other way around!).

3.2. Defensive Recommendations

Observations on the IDS system itself

While it is certainly good to have a network intrusion detection system in place, it should also be kept up to date with the latest software and configuration. This is exceedingly true for signature based ones as Snort [0] (its ability to detect attacks is similar to the antivirus products available today; the virus definitions must be kept up to date, or it will miss new viruses). Hence, my first recommendation is to upgrade the NIDS sensor itself.

After the upgrade, the rules should be customised to reduce the vast amounts of false positives. A single SSH session generated approximately 4500 alerts due to the fact the source port of the client happened to be 65535. Those over-generalised rules may cause more problems than they solve.

It is essential that the alert load is such that the analysts can handle it. An analyst that is drowned in false alerts is likely to miss out potentially important ones.

Furthermore, from the fact that I have seen no alerts or scans from internal to other internal systems, I surmise that the sensor is located at the border. It is possible that I have only been given data from one of several sensors; but if not (i.e. this is the only one), I highly recommend that an internal sensor between key networks is considered.

Network observations

Even though many universities do not deploy firewalls by policy or other reasons, I still suggest that a firewall is considered. The load on the NIDS system would be heavily reduced if it were placed behind a filtering device. Additionally, it could protect the organisation from bad reputation or in worst case liability caused by outsiders using internal hosts as a springboard for further attacks.

If my analysis regarding the placement of the NIDS sensor is correct, the internal routers should be investigated for their policies regarding RFC 1918 [33] and RFC 3330 [34] addresses (private and link local addresses) as these have no business being routed to the outside.

Policy issues

As I have no knowledge of the site's security policy or acceptable use policies, I will say very little about this subject. The usual apply, though. If there are no policies regarding acceptable use in place (from the traffic being observed, particularly policies relating to scanning and file sharing would be of interest), such documents should be written, approved and clearly communicated to the users of the network.

Suspicious internal systems

During the assessment, some internal systems were found performing suspicious activities. It would be prudent to further investigate these systems.

MY.NET.84.235

This system should be investigated as soon as possible as it appears to be responsible for multiple activities of questionable nature:

- File sharing with Overnet/Edonkey2000 [35]
- Scanning for other users with Overnet/Edonkey2000
- Connections to external backdoors left by Red Worm/Adore [36]
- Possibly infected by SdBot (or similar) backdoor [37]
- It is probably running some sort of web server

Systems that are likely to be infected by worms

The below systems are probably infected by an Agobot/Gaobot/PolyBot variant:

- MY.NET.70.96
- MY.NET.42.2
- MY.NET.153.174
- MY.NET.97.66
- MY.NET.112.152
- MY.NET.151.75
- MY.NET.43.10
- MY.NET.150.199
- MY.NET.66.56
- MY.NET.80.224
- MY.NET.80.5

Port 135 scanning

The hosts MY.NET.111.51 and MY.NET.81.39 performed massive scanning for external systems listening on port 135.

Miscellaneous UDP scanning from MY.NET.153.53

The system targeted port 1214 on 4488 different hosts, as well as port 65535 on 5 hosts, among them 24.5.46.4. All traffic used UDP.

Systems trying to enumerate NetBIOS shares on Link Local addresses

While this traffic could be benign, I still recommend that some investigation is performed as to locate the reason for the traffic.

- MY.NET.11.4
- MY.NET.11.5
- MY.NET.11.6
- MY.NET.11.7

Systems performing external NetBIOS share enumeration

- MY.NET.150.44
- MY.NET.150.198
- MY.NET.75.13

Troublesome external hosts

24.5.46.4 (c-24-5-46-4.client.comcast.net)

Several internal hosts connected to this host on TCP port 65535 (which may be associated with the Red Worm/Adore backdoor). Yet another internal host connected to it on the same port, but over UDP. It may be that some other service is running there. Further investigation is recommended.

199.131.21.34 (zc7831522.ip.fs.fed.us)

The network segment appears to belong to the US Department of Agriculture. The host in question appears to be infected by one worm or another; it scanned and tried to exploit multiple internal hosts.

68.55.192.251 (pcp229440pcs.catonv01.md.comcast.net)

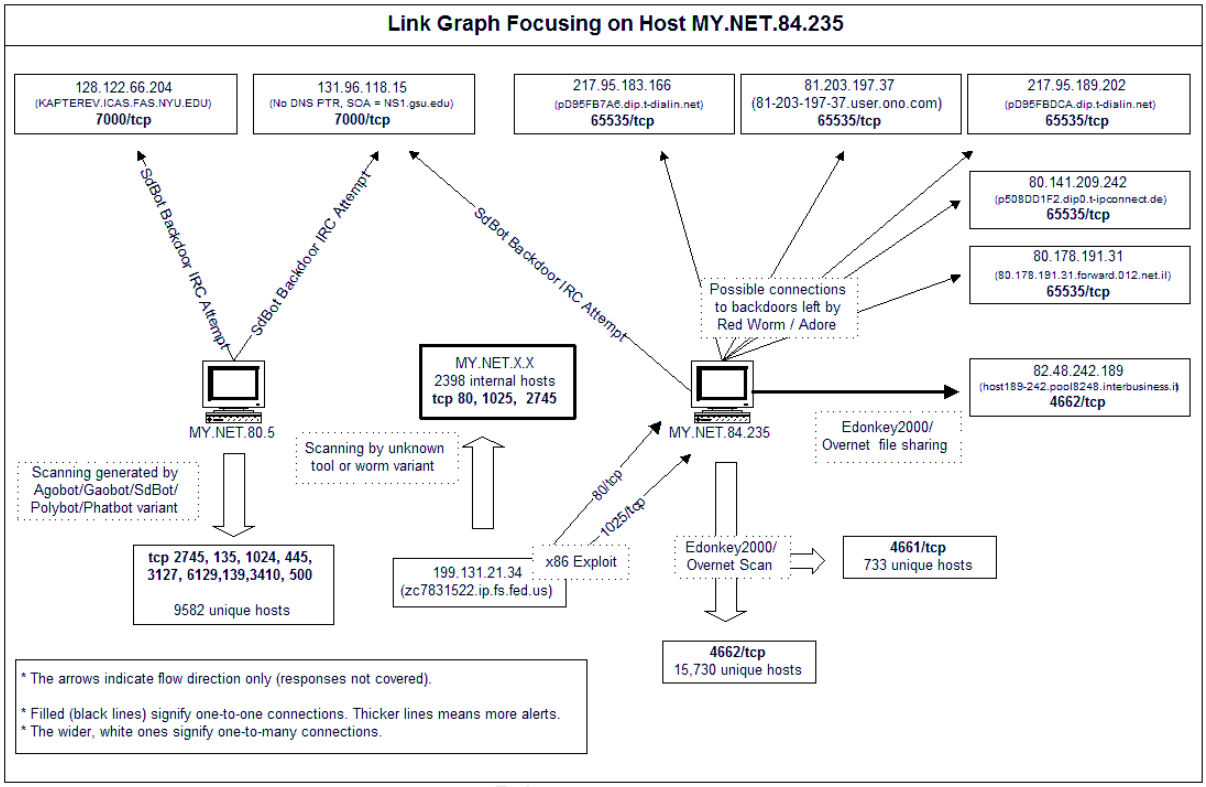
Traces indicate that this system connected to a VNC service located on MY.NET.111.51. It may be of interest to investigate this further, especially since MY.NET.111.51 did participate in other suspicious activity (RPC scanning).

128.122.66.204 (KAPTEREV.ICAS.FAS.NYU.EDU)

This is a system to which 16 internal hosts generated traffic that may very well be from a SdBot backdoor. Totally 104 matching packets were sent to it from the 16 hosts (each alert was part of a separate session) .

3.3. Link Graph

The graph below represents relationships with one of the more troublesome internal hosts – MY.NET.84.235.



3.4. List of Analysed Files

The files listed below were retrieved from the http://isc.sans.org/logs/ site [38] for analysis. Note that the date sequence in the file names differ for the OOS files. For no reason I can understand, there seem to be some inconsistencies in the naming process; however the data inside the files cover the same dates (040407-040411). The contents also reflects the date the file was created on the website.

"Alert" files	*"Out of Spec" files*	*"Scan" files*
alert.040407.gz	oos_report_040403	scans.040407.gz
alert.040408.gz	oos_report_040404	scans.040408.gz
alert.040409.gz	oos_report_040405	scans.040409.gz
alert.040410.gz	oos_report_040406	scans.040410.gz
alert.040411.gz	oos_report_040407	scans.040411.gz

An important statement must be made regarding the contents in the scans file. In difference with the alert and oos files, internal IP addresses were *not* obfuscated. I have been able to correlate traffic patterns from the other two files to establish the real numbers of MY.NET. While this may be intentional, and the MY.NET substitutions may only be for the purpose to distinguish what the internal network was, I have replaced the two octets in the scans file to MY.NET as well. Except for protecting the information, it also gives the advantage of making the report more consistent. A side effect is that I will not list any reverse DNS information for internal hosts, as this would be an obvious give-away.

3.5. Top Talkers

My criteria for choosing the Top Talkers was simply to select the ten most frequent source addresses of each log type (scans, alerts and OOS). As I will refer back to these lists in the rest of the report, I will call the scan list T10S, the alert list T10A, and the OOS list T10O.

Top Ten Scanners

Top Ten Scan Sources (T10S)		
No	***IP Address***	***Number of Packets***
1	MY.NET.1.3	2,890,566
2	MY.NET.111.51	1,623,412
3	MY.NET.153.35	1,522,557
4	MY.NET.81.39	1,189,565
5	MY.NET.70.96	1,130,812
6	MY.NET.112.152	1,082,026
7	MY.NET.1.4	796,703
8	MY.NET.66.56	338,680
9	MY.NET.84.235	295,233
10	MY.NET.42.2	253,157

Top Ten Attackers

Top Ten Attack Sources (T10A)			
No	***IP Address***	***Resolves To***	***Count***
1	212.76.225.24	cable-212.76.225.24.coditel.net	7,559
2	MY.NET.11.7	<omitted>	7,017
3	MY.NET.84.235	<omitted>	3,952
4	199.131.21.34	zc7831522.ip.fs.fed.us	3,480
5	68.81.0.87	pcp01333933pcs.columb01.pa.comcast.net	2,993
6	141.157.102.155	pool-141-157-102-155.balt.east.verizon.net	2,693

Top Ten Attack Sources (T10A)			
7	MY.NET.60.16	<omitted>	2,169
8	131.92.177.18	aeclt-cf00a4.apgea.army.mil	2,166
9	68.57.90.146	pcp912734pcs.brndml01.va.comcast.net	1,660
10	69.138.77.62	pcp08479849pcs.desoto01.md.comcast.net	1,628

Top Ten Generators of Anomalous Data (OOS)

Top Ten OOS Sources (T10O)			
No	***IP Address***	***Resolves To***	***Count***
1	68.54.84.49	pcp01741335pcs.howard01.md.comcast.net	1,716
2	202.144.28.167	kurinji.tenet.res.in	699
3	141.224.64.4	bessie.augsburg.edu	265
4	62.174.236.17	FW-17-236.go.retevision.es	171
5	193.170.194.27	<not available>	169
6	66.225.198.20	unknown.splashhost.net	166
7	80.54.249.136	cnet-u-249-136.petrus.pl	163
8	80.38.206.68	68.Red-80-38-206.pooles.rima-tde.net	116
9	MY.NET.199.202	<omitted>	96
10	202.54.60.162	<not available>	90

3.6. Registry Information

I have selected the (in my opinion) five most suspicious external hosts and retrieved network registry information for them. The result is displayed in this section. They are ordered by priority, with the most suspicious system first.

199.131.21.34 (zc7831522.ip.fs.fed.us)

The host (number 4 on Top Ten Attackers) appears to be sending shellcode typically associated with buffer overflows to multiple internal hosts. It also performed broad-scale scanning indicating a possible worm infection.

Registry: ARIN

```
OrgName:    USDA Office of Operations
OrgID:      UOO-2
Address:    Suite 133, Building A
Address:    2150 Centre Ave
City:       Fort Collins
StateProv:  CO
PostalCode: 80526
```

```
Country:    US

NetRange:   199.128.0.0 - 199.159.255.255
CIDR:       199.128.0.0/11
NetName:    USDA-CBLK
NetHandle:  NET-199-128-0-0-1
Parent:     NET-199-0-0-0-0
NetType:    Direct Allocation
NameServer: NS.USDA.GOV
NameServer: NS2.USDA.GOV
NameServer: NS3.USDA.GOV
Comment:
RegDate:    1994-02-08
Updated:    2000-06-16

TechHandle: ZU20-ARIN
TechName:   USDA - Office of the ChiefInformation Officer
TechPhone:  +1-970-295-5277
TechEmail:  Network.Operations@usda.gov

OrgAbuseHandle: ZU20-ARIN
OrgAbuseName:   USDA - Office of the ChiefInformation Officer
OrgAbusePhone:  +1-970-295-5277
OrgAbuseEmail:  Network.Operations@usda.gov

OrgNOCHandle: ZU20-ARIN
OrgNOCName:   USDA - Office of the ChiefInformation Officer
OrgNOCPhone:  +1-970-295-5277
OrgNOCEmail:  Network.Operations@usda.gov

OrgTechHandle: ZU20-ARIN
OrgTechName:   USDA - Office of the ChiefInformation Officer
OrgTechPhone:  +1-970-295-5277
OrgTechEmail:  Network.Operations@usda.gov
```

128.122.66.204 (KAPTEREV.ICAS.FAS.NYU.EDU)

This is a system to which 16 internal hosts generated traffic that may very well be from a SdBot backdoor. Totally 104 matching packets were sent to it from the 16 hosts (each alert was part of a separate session) .

Registry: ARIN

```
OrgName:    New York University
OrgID:      NYU
Address:    Academic Computing Facility
Address:    251 Mercer Street
City:       New York
StateProv:  NY
PostalCode: 10012
Country:    US

NetRange:   128.122.0.0 - 128.122.255.255
CIDR:       128.122.0.0/16
NetName:    NYU-NET
NetHandle:  NET-128-122-0-0-1
Parent:     NET-128-0-0-0-0
NetType:    Direct Assignment
NameServer: CMCL2.NYU.EDU
```

```
NameServer: EGRESS.NYU.EDU
NameServer: NYUNSB.NYU.EDU
Comment:
RegDate:    1986-05-02
Updated:    2001-05-21

TechHandle: ZN68-ARIN
TechName:   New York University
TechPhone:  +1-212-998-3431
TechEmail:  NOC@nyu.edu
```

24.5.46.4 (c-24-5-46-4.client.comcast.net)

Multiple internal hosts connected to port 65535 on this system. As the port is not usually associated with any benign traffic, it became exceedingly interesting to pursue this further.

Registry: ARIN

```
CustName:   Comcast Cable Communications
Address:    3 Executive Campus
Address:    5th Floor
City:       Cherry Hill
StateProv:  NJ
PostalCode: 08002
Country:    US
RegDate:    2003-11-05
Updated:    2003-11-05

NetRange:   24.4.0.0 - 24.5.255.255
CIDR:       24.4.0.0/15
NetName:    BAYAREA-9
NetHandle:  NET-24-4-0-0-1
Parent:     NET-24-0-0-0-1
NetType:    Reassigned
Comment:    NONE
RegDate:    2003-11-05
Updated:    2003-11-05

OrgAbuseHandle: NAPO-ARIN
OrgAbuseName:   Network Abuse and Policy Observance
OrgAbusePhone:  +1-856-317-7272
OrgAbuseEmail:  abuse@comcast.net

OrgTechHandle: IC161-ARIN
OrgTechName:  Comcast Cable Communications Inc
OrgTechPhone: +1-856-317-7200
OrgTechEmail:  cips_ip-registration@cable.comcast.com
```

212.76.225.24 (cable-212.76.225.24.coditel.net)

Number one of the Top Ten Attackers. Suspicious traffic from it indicates a possible attack against security systems such as IDSses and firewalls as well as TCP/IP stack fingerprinting.

Registry: RIPE

```
inetnum:      212.76.225.0 - 212.76.225.255
netname:      CODITEL
descr:        Coditel - Internet Services
country:      BE
admin-c:      XD6
tech-c:       YB490-RIPE
status:       ASSIGNED PA
notify:       tech.registry@coditel.be
mnt-by:       CODITEL-MNT
mnt-lower:    CODITEL-MNT
changed:      xavier.darche@coditel.be 20010109
changed:      xavier.darche@coditel.be 20030513
changed:      xavier.darche@coditel.be 20030514
source:       RIPE

route:        212.76.224.0/19
descr:        DN-21276224-BLOCK
descr:        Coditel
descr:        For routing issues:noc@attglobal.net
remarks:      ----------------------------------------
remarks:      Abuse notifications to: abuse@coditel.be
remarks:      ----------------------------------------
origin:       AS2686
notify:       xdarche@netscape.net
mnt-by:       CODITEL-MNT
changed:      xavier.darche@coditel.be 20000605
changed:      xavier.darche@coditel.be 20030514
source:       RIPE

person:       Xavier Darche
address:      Coditel SA
address:      26 Rue des Deux Eglises
address:      B - 1000 Brussels
address:      Belgium
phone:        +32 2 226 54 04
fax-no:       +32 2 218 77 00
e-mail:       xavier.darche@coditel.be
nic-hdl:      XD6
remarks:      ----------------------------------------
remarks:      Abuse notifications to: abuse@coditel.be
remarks:      ----------------------------------------
notify:       xavier.darche@coditel.be
mnt-by:       CODITEL-MNT
changed:      xavier.darche@coditel.be 20030416
source:       RIPE

person:       Yves Beckers
address:      Coditel SA
address:      26 Rue des Deux Eglises
address:      B - 1000 Brussels
address:      Belgium
phone:        +32 2 226 54 23
fax-no:       +32 2 219 77 25
e-mail:       yves.beckers@coditel.be
nic-hdl:      YB490-RIPE
remarks:      ----------------------------------------
remarks:      Abuse notifications to: abuse@coditel.be
remarks:      ----------------------------------------
notify:       yves.beckers@coditel.be
```

```
mnt-by:       CODITEL-MNT
changed:      xavier.darche@coditel.be 20030416
source:       RIPE
```

141.157.102.155 (pool-141-157-102-155.balt.east.verizon.net)

The host 141.157.102.155 initiated a 75 minutes long SSH session to the internal host MY.NET.60.16. It is of interest to see what organisation the host is associated with. The system is number 6 on the Top Ten Attackers list.

Registry: ARIN

```
CustName:   Verizon Internet Services
Address:    1880 Campus Commons Drive
City:       Reston
StateProv:  VA
PostalCode: 20191
Country:    US
RegDate:    2002-03-21
Updated:    2002-03-21

NetRange:   141.157.57.0 - 141.157.126.255
CIDR:       141.157.57.0/24, 141.157.58.0/23, 141.157.60.0/22,
            141.157.64.0/19, 141.157.96.0/20, 141.157.112.0/21,
            141.157.120.0/22, 141.157.124.0/23, 141.157.126.0/24
NetName:    VZ-DSLDIAL-CYVLMD-9
NetHandle:  NET-141-157-57-0-1
Parent:     NET-141-149-0-0-1
NetType:    Reassigned
Comment:
RegDate:    2002-03-21
Updated:    2002-03-21

TechHandle: ZV20-ARIN
TechName:   Verizon Internet Services
TechPhone:  +1-703-295-4583
TechEmail:  noc@gnilink.net

OrgAbuseHandle: VISAB-ARIN
OrgAbuseName:   VIS Abuse
OrgAbusePhone:  +1-703-295-4583
OrgAbuseEmail:  abuse@verizon.net

OrgTechHandle: ZV20-ARIN
OrgTechName:   Verizon Internet Services
OrgTechPhone:  +1-703-295-4583
OrgTechEmail: noc@gnilink.net
```

3.7. Activity originating from internal hosts (MY.NET.X.X)

Attacks and other suspicious activity originating from the internal network could imply compromised hosts being used by a malicious outsider as well as illicit activity from an internal party. As such, internal activity is given extra consideration. The starting point are those internal hosts found on the T10A list. Despite this, the data

presented will be organised in terms of events rather than hosts as this will give a better overview.

SMB Name Wildcards

The first system to take a closer look at is MY.NET.11.7. (number 2 on T10A) It is responsible for 7017 alerts, all of them being of type "*SMB Name Wildcard*". This could potentially be a information gathering attack (the "nbtstat -A" command will generate this alert). For more information regarding the signature, see the following URL at www.whitehats.com:

http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids177&view=research [39]

To fully analyse this event, it becomes interesting to check other hosts and destinations for the same attack signature. The data is presented below.

SMB Wildcards to Link Local addresses		
Source	***Target***	***Count***
MY.NET.11.7	169.254.0.0	5,186
MY.NET.11.7	169.254.25.129	1,831
MY.NET.11.6	169.254.0.0	518
MY.NET.11.5	169.254.0.0	19
MY.NET.11.4	169.254.25.129	17

In addition, 111 different hosts on other internal networks than MY.NET.11.0/24 sent this traffic to 169.254.45.176 for a total of 802 times.

What is peculiar with the traffic is that the 169.254 network is an IANA "special allocation". From RFC 3330 [34]:

> *169.254.0.0/16 - This is the "link local" block. It is allocated for communication between hosts on a single link. Hosts obtain these addresses by auto-configuration, such as when a DHCP server may not be found.*

The only clients I am aware of that actually do auto-configuration when failing to get a DHCP response is Microsoft Windows workstations.

From reading older reports[32] concerning this network, I have come to the conclusion that MY.NET.11.6 and MY.NET.11.7 may be Windows domain controllers. Then it is also logical that the other systems on the network is Windows servers. Thus, a worm striking one of them might as easy strike the others (it is not uncommon that all servers on a small segment are patched at the same time. It then follows that if one system is vulnerable, chances are that all of them are).

32 Tod Beardsley's GCIA practical is one such example [40].

There is no exact pattern to the packets, but they are evenly spread throughout the complete five day dataset.

It is somewhat puzzling why the traffic is occurring. One explanation could be that the sending hosts are infected with one of the numerous worms that scans for open Windows shares to propagate through (a worm would probably not know or care about the special allocation of 169.254.0.0/16). However, this is a wild guess. In any case, these systems should be looked into to determine what is going on.

Additionally, the following systems have issued wildcard requests to a number of different unique external hosts. While this could be worm activity, I think it to be more likely they are scanning these hosts for other purposes.

SMB Wildcards from internal hosts (NetBIOS enumeration)	
Source	***Unique Targets***
MY.NET.75.13	171
MY.NET.150.198	167
MY.NET.150.44	165

"DDOS mstream" alerts

From MY.NET.84.235 (number 3 on T10A), there are a total of 3248 occurrences of the alert "DDOS mstream handler to client". The alleged mstream traffic is directed to 6 different hosts. The Snort rule uses both port and content. The content check reduces the likelihood for false positives. According to the Snort rule documentation, there is no known false positives for the rule. Unfortunately, the content check is a bit too vague; a ">" in the payload in addition to the port (either one of 12754 and 15104) will trigger the alert.

The table below shows the distribution of the alerts for this host.

"DDOS mstream handler to client" alerts from MY.NET.84.235				
SRC Port	***Target***	***Resolves To***	***DST Port***	***Count***
12754	82.48.242.184	host184-242.pool8248.interbusiness.it	4662	3,240
12754	62.42.66.52	MS1-3A-u-0563.mc.onolab.com	4662	3
15104	81.102.85.92	cpc1-cosh5-4-0-cust92.cos2.cable.ntl.com	4662	2
15104	81.69.163.174	amr-dmr-185ae.adsl.wanadoo.nl	4662	1
15104	217.236.97.47	pD9EC612F.dip.t-dialin.net	4662	1
15104	80.15.47.94	ARouen-106-1-10-94.w80-15.abo.wanadoo.fr	4662	1

My analysis is that this is a false positive (no matter what the Snort documentation says). For a real mstream handler, I would expect more hosts, and not 3240 commands to one of them and just an occasional one for the others. What I think generated this traffic is the file sharing clients Edonkey2000 and Overnet.

Both clients are made by the same company, and both makes use of destination port 4662.

From the Edonkey FAQ [41]:

> ***What ports does the donkey use?***
> *TCP port 4662 to connect to other clients.*

From the Overnet FAQ [42]:

> ***What ports does Overnet use and how do I change them?***
> *It can run over any port. It uses TCP port 4662 by default to connect to other clients.*

After all, source ports 12754 and 15104 are perfectly valid ephemeral ports. And it is not very unlikely that a file containing one or more ">" would be transferred by a file sharing protocol. For example, most MP3 files seem to contain far more than one ">". And if popular rumours are to be believed, MP3's are commonly swapped through such systems.

Another probable false positive is a connection from MY.NET.60.17 to 65.54.252.99 on port 25. This looks like SMTP to me; especially so since the destination resolves to mc5.bay6.hotmail.com.

High port 65535 tcp - possible Red Worm - traffic

The Snort signature that generated this alert is not aimed to detect the worm itself, but rather the backdoors left in place by it. Worth to mention that the worm made more fame under the name "Adore", for which Red Worm is an alias.

According to F-Secure (http://www.f-secure.com/v-descs/adore.shtml): [36]

> *The script also replaces "/sbin/klogd" with a version that has a backdoor. The backdoor activates when it receives a ping packet with correct size, and opens a shell in the port 65535. Orginal "klogd" will be saved to "/usr/lib/klogd.o".*

The Snort rule is far from perfect. It triggers on all TCP or UDP packets where either the source or destination port is 65535. That is an ephemeral port, meaning that most systems sooner or later is bound to use it as source port when initiating new sessions.

For example, there are such alerts from MY.NET.60.16 (number 7 on T10A), apparently targeting 141.157.102.155 (number 6 on the same list). During a time period of 75 minutes, 2168 alerts were generated, and all of them had the same source and destination ports. The source port in question was 22. Also, another 2693 alerts were generated, but this time 141.157.102.155 was the source, and MY.NET.60.16 the destination. The time interval and port number was identical. To have a rule trigger on every packet in both directions is impractical in my opinion. My

conclusion is that these 4861 alerts were caused by a single SSH session. To back up this analysis, I verified that the first packet with destination port 22 preluded the first one with source port 22 (approximately 0.02 seconds between them).

That an external host established a SSH session (since it lasted for over an hour, it can be assumed successful authentication took place) to an internal one may be cause of concern, or be perfectly legitimate, all depending on the site's policy.

The rule is not entirely useless, though. If all connections where one of ports belongs to a well-known service are removed from the search criteria, a few hosts remains which may be worth to look further into. The ports removed were 22, 25, 80, 110, 143 and 443.

The first table shows a pattern where several internal hosts connected to a single destination host ("sessions" indicate how many unique source ports were used, but not how much data passed through each connection).

Port 65535 connections to 24.5.46.4 (c-24-5-46-4.client.comcast.net)	
Source	***Number of Sessions***
MY.NET.97.51	112
MY.NET.97.92	47
MY.NET.97.182	36
MY.NET.97.196	30
MY.NET.97.104	21
MY.NET.97.124	4
MY.NET.98.87	4

The second table shows a case where one internal host connected to multiple external hosts.

Port 65535 connections from MY.NET.84.235		
Target	***Resolves To***	***Number of Sessions***
81.203.197.37	81-203-197-37.user.ono.com	70
217.95.183.166	pD95FB7A6.dip.t-dialin.net	36
217.95.189.202	pD95FBDCA.dip.t-dialin.net	5
80.141.209.242	p508DD1F2.dip0.t-ipconnect.de	3
80.178.191.31	80.178.191.31.forward.012.net.il	2

Patterns like above generally makes me twitchy; something is probably going on here. It could be the Red Worm (Adore) backdoor, but it could also be something else (I have lost count of the number of file sharing and instant messaging programs out there, and everyone seem to be configurable to run on some custom port or another). Another thing worth to mention is that the number of packets per session is

quite low; most are between 1 and 20 (this is one direction only, but as it is TCP, the number on the other side should be approximately the same).

On a final note - there is no registered (i.e. by IANA) usage of port 65535; neither does it seem to be any well-known but not registered application making use of it.

3.8. Activity from external systems

"MY.NET.30.X Activity"

Apparently, there are rules that logs some - or all -traffic with a destination of MY.NET.30.3 or MY.NET.30.4. This being the case, one can only surmise those systems are considered important. As both events occur frequently both in terms of total events and in the event set of the most frequent external systems, I feel it necessary to present what the traffic constituted.

The four most frequent “offenders” are also on the T10A list:

Most frequent offenders in terms of events			
T10A	***Source Address***	***Resolves To***	***Most Targeted Port***
5	68.81.0.87	pcp01333933pcs.columb01.pa.comcast.net	51443
8	131.92.177.18	aeclt-cf00a4.apgea.army.mil	524
9	68.57.90.146	pcp912734pcs.brndml01.va.comcast.net	524
10	69.138.77.62	pcp08479849pcs.desoto01.md.comcast.net	524

If we change view and look at the sessions with most packets (assuming that source and destination port is the same during a session, and that all packets - i.e. not only SYN - were logged), we get a pretty accurate picture of what is going on.

Sessions going to MY.NET.30.3				
Source IP	***Resolves To***	***SRC Port***	***DST Port***	***Count***
131.92.177.18	aeclt-cf00a4.apgea.army.mil	1033	524	2166
68.57.90.146	pcp912734pcs.brndml01.va.comcast.net	1030	524	1136
68.55.113.194	pcp311543pcs.woodln01.md.comcast.net	32852	524	719
...and so on down to:				
68.43.170.140	bgp01087647bgs.waren301.mi.comcast.net	1880	80	28
68.43.170.140	bgp01087647bgs.waren301.mi.comcast.net	1670	80	21
207.245.65.114	207-245-65-114.theenterprisecenter.com	51601	80	16

The protocol associated with port 80 is known to most people. The conclusion that a web server is running on the system seem natural. Port 524 may not be as common, but in this case, most of the traffic goes there. People with a background in Novell

system will immediately think "NCP" or NetWare Core Protocol. A brief explanation can be found at protocols.com [43]:

> *The Novell NetWare Core Protocol (NCP) manages access to the primary NetWare server resources. It makes procedure calls to the NetWare File Sharing Protocol (NFSP) that services requests for NetWare file and print resources.*

The system should either be a Novell NetWare server running a web server, or a Linux system with a web server and NCP services. Both seem as likely to me.

Sessions going to MY.NET.30.4 (each row indicates a session)				
Source IP	***Resolves To***	***SRC Port***	***DST Port***	***Count***
68.81.0.87	pcp01333933pcs.columb01.pa.comcast.net	43461	51443	1,417
68.81.0.87	pcp01333933pcs.columb01.pa.comcast.net	51652	51443	642
68.55.116.84	pcp312201pcs.woodln01.md.comcast.net	41413	524	146
68.81.0.87	pcp01333933pcs.columb01.pa.comcast.net	39108	51443	125
68.67.86.74	pa-jefferson1a-74.chvlva.adelphia.net	1028	524	40
69.140.109.252	pcp04418584pcs.nrockv01.md.comcast.net	1082	80	38
68.57.90.146	pcp912734pcs.brndml01.va.comcast.net	1033	524	32
172.169.96.44	ACA9602C.ipt.aol.com	1978	80	31
138.88.183.54	pool-138-88-183-54.res.east.verizon.net	50410	51443	31
69.137.151.85	pcp08726942pcs.towson01.md.comcast.net	1204	80	28

The same analysis as for MY.NET.30.3 applies here, as both port 80 and 524 is targeted. Interestingly, another port takes most of the traffic: 51443. That one is not registered by IANA. So what is it?

By all probability, it is a service for Novell servers called NetStorage. It runs on two ports, 51080 for HTTP, and 51443 for HTTPS. More information about this can be found at the URL below [44]:

> http://developer.novell.com/research/appnotes/2002/june/03/a0206033.htm

As it is a network based storage system, it is not too extraordinary that it receives more traffic than other ports.

Activity from 199.131.21.34 (zc7831522.ip.fs.fed.us)

This system is number 4 on T10A.

One alert initially (more about this later) appears as a bit misguided; it is a single packet triggering a "MY.NET.30.4 activity" alert towards port 2745. The remaining 3479 attacks is "EXPLOIT x86 NOOP", which could be shellcode associated with

buffer overflow attempts. The destination ports are port 80 (2885 occurrences) and port 1025 (594 occurrences). I am usually a bit sceptical against x86 NOP alerts targeting port 80, as any downloaded GIF-image will match the signature as well.

However, port 80 false positives generally tend to be concentrated on a single web server. In this case, there is probably genuine cause for concern. In total, 436 unique hosts on MY.NET were targeted on port 80, and 75 hosts on port 1025 (in total, 499 unique hosts were attacked, meaning a few systems were attacked on both ports).

Unfortunately, it is quite difficult to determine whether any attack was successful. What I will do is to list the top five destinations in terms of alerts (the majority were targeted one or two times).

Top destinations for port 80			***Top destinations for port 1025***	
Destination	***Count***		***Destination***	***Count***
MY.NET.84.235	773		MY.NET.84.227	31
MY.NET.84.236	767		MY.NET.84.203	28
MY.NET.84.204	305		MY.NET.84.176	24
MY.NET.84.135	190		MY.NET.84.187	24
MY.NET.70.148	9		MY.NET.84.205	24

MY.NET.84.235 deserves special consideration, as it already has been listed as a potential problem. In fact, with a count of 18, it has a higher score than average on port 1025 too.

To be able to attack only hosts listening on the target ports, it is realistic to assume some type of scanning preluded the attacks. In this case, a SYN scan swept over 2398 internal hosts.

Distribution of scans from 199.131.21.34 (zc7831522.ip.fs.fed.us)		
Destination Port	***Probable Target Service***	***Count***
2745	Backdoor left by Bagle	4422
1025	Buffer Overflow (MS03-032)	3602
80	Buffer Overflow (MS03-032)	1579

The above information explains the odd alert (MY.NET.30.4 Activity) previously mentioned. The rule picked up on the scan rather than the attack.

The scan does not appear to be symmetric. By that I mean that all ports were not scanned on every host. This asymmetry could indicate a worm, but it is not impossible that the host in question has been “manually” compromised and is used as a springboard for various malicious activity.

On a final note, the source of the attack originates from a netblock owned by the US Department of Agriculture. They may be more interested than the average cable provider in knowing that one of their hosts are participating in illicit activity.

Activity from 212.76.225.24 (cable-212.76.225.24.coditel.net)

All systems on the T10A list have been dealt with except one: the most frequent offender of them all. So what did this system do?

Overview of activity from 212.76.225.24 (cable-212.76.225.24.coditel.net)		
Attack	***Target***	***Count***
Tiny Fragments - Possible Hostile Activity	MY.NET.43.3	7,487
Null scan!	MY.NET.43.3	57
Tiny Fragments - Possible Hostile Activity	MY.NET.43.2	15

The majority of the "*Tiny Fragments*" traffic occurred between 00:30 and 08:00 during the 10th of April 2004. The 15 packets directed to MY.NET.43.2 happened between 05:29 and 08:58 one day later (April 11). All packets had a recorded source and destination port of zero.

The "*Null scan!*" triggers on all TCP packets that have no TCP flags at all. Of these, most have source and destination ports of zero. The remaining 16 appears to be random.

I can think of two reasons why the traffic is occurring. The first one is a broken network device somewhere. The second one is traffic designed to cause a denial of service (strange fragmentation generally makes me suspicious) against a NIDS, firewall, or the targeted host itself. As it is best to assume the worst until the opposite is proved, I will suspect the latter.

The system is also a perpetrator of some peculiar scans, all of them targeting MY.NET.43.3. There is a total of 118 scans. Among these, 27 different combination of TCP flags can be found, which is quite impressive. Most of the combinations are invalid (one such example is a packet with URG, ACK, SYN and RST).

This scanning may be part of the other attack, but may also indicate an attempt to fingerprint the IP stack of the target.

The system does not occur in the OOS files, which I almost would have expected by seeing a bunch of very weird packets.

3.9. More on Scans

Except for the scans that has already been discussed, there are some additional information that deserves to be mentioned. It is a bit alarming that the top ten

scanning systems are internal hosts targeting external ones. The relationship between internal and external hosts are shown below:

- Totally scanned hosts 3,998,813
- Totally scanned external hosts: 3,983,071
- Totally scanned internal hosts 15,742
- Percent internal hosts scanned: 0,4%

MS-RPC scanning

Two of the systems were found to be scanning large number of hosts on port 135, which is typically associated with MS-RPC services. In both cases, the scanning was incremental in terms of destination and source port.

Scans targeting port 135 TCP		
T10S	***Source***	***Unique targets***
2	MY.NET.111.51	1,622,989
4	MY.NET.81.39	1,188,983

The problem is, there are so many attacks associated with port 135 that it is hard to tell what the purpose is. It may, for example, be a reconnaissance scan, storing the results for future use. There is no attacks corresponding to this scanning activity (this does not mean there weren't any, only that none were recorded).

Although not directly relevant to this activity, it should be mentioned that MY.NET.111.51 does appear in the alert logs. On three occasions, the signature "*RFB - Possible WinVNC - 010708-1*" triggered. It went off on different days (9th, 10th, and 11th of April), and different destination ports were used (source port was 5900). This may indicate that the system 68.55.192.251 did connect to the internal host using VNC, a remote desktop control utility which is available free of charge.

File sharing client scanning from MY.NET.84.235

One host that has been mentioned many times before, and will be mentioned again is MY.NET.84.235. It earned position 9 on the T10S list by scanning 15,730 unique targets on port 4662, which I previously stated is likely to belong to the Edonkey2000 or Overnet file sharing clients.

Extensive DNS utilisation

A vast number of scans were recorded with a target port of 53 (UDP). Further investigation shows that there is two internal systems that generated this traffic.

Scans targeting port 53 UDP			
T10S	***Source***	***Unique targets***	***Number of packets***
1	MY.NET.1.3	103,753	2,878,847
7	MY.NET.1.4	58,528	786,975

There is also a few TCP packets on port 53:

Scans targeting port 53 TCP			
T10S	***Source***	***Unique targets***	***Number of packets***
1	MY.NET.1.3	29	300
7	MY.NET.1.4	7	46

When the traffic also is sorted on destination hosts, the picture starts to emerge.

Traffic from MY.NET.1.3		
Target	***Resolves To***	***Number of packets***
69.6.57.7	<not available>	82,845
69.6.57.9	<not available>	82,735
192.26.92.30	c.gtld-servers.net	47,847
69.6.57.8	<not available>	40,593
69.6.57.10	<not available>	40,501

Traffic from MY.NET.1.4		
Target	***Resolves To***	***Number of packets***
192.26.92.30	c.gtld-servers.net	20,655
192.48.79.30	j.gtld-servers.net	15,120
192.5.6.30	a.gtld-servers.net	13,071
69.6.25.84	n/a, SOA = ns1.wholesalebandwidth.com	11,790
65.244.133.127	n/a, SOA = auth00.ns.uu.net	11,116
65.244.133.129	n/a, SOA = auth00.ns.uu.net	10,472

It seems like MY.NET.1.3 and MY.NET.1.4 are DNS servers serving internal hosts. In order words, I would conclude that this scanning is in fact false positives (i.e. it is normal traffic). Les Gordon's practical [45] confirms my theory about these systems being DNS servers.

When comparing the internal sources with recorded alerts, all[33] we get is three Red Worm false positives, when source port is 65535, but destination still is 53 TCP.

33 To be truly honest, MY.NET.1.3 has a single Tiny Fragment alert associated with it as well, going to 216.127.235.73, with source and destination ports of 0. As this is an isolated event, it may be nothing more than a corrupt packet or even a problem with the Snort sensor itself.

From looking at the timestamps, it appears to be one attempt and two retries, most likely caused by a truncated answer on an UDP query together with a firewall blocking TCP port 53 at the other end.

The four servers on the same network (69.6.57.0) is a bit interesting. One would think that DNS servers would have valid reverse information. The funny thing is that there are normal DNS records for them:

nserver: a.ns.alwaysclickingonemails.com 69.6.57.7
nserver: b.ns.alwaysclickingonemails.com 69.6.57.8
nserver: c.ns.alwaysclickingonemails.com 69.6.57.9
nserver: d.ns.alwaysclickingonemails.com 69.6.57.10

Source: http://www.dslreports.com/forum/remark,10137492~mode=flat [46]

The netblock belongs to WholesaleBandwidth, Inc.

Apparently, both the netblock and the domain is associated with excessive UCE (Unsolicited Commercial Email, or "spam", which it is more frequently known as). If the internal email servers do a reverse lookup on incoming connections (which is fairly common practice), it would certainly explain why its DNS servers get a lot of traffic (approximately the same as the gTLD servers).

Miscellaneous UDP scanning from MY.NET.153.35

MY.NET.153.35 (position 3 on the T10S) performed a wide variety of scans. In total, 1,522,557 packets were recorded. Most of them were UDP. The most frequent grouping targeted port 1214/udp on 4488 different hosts, for a total of 19,494 packets. Except from that, I have not perceived any particular patterns.

When running the source against the attack data, I found at least some alerts that may warrant a closer inspection on this host. The interesting alert is of type "*High port 65535 udp - possible Red Worm – traffic*", which occurs more frequently as its TCP counterpart. In this case, all packets has a source port of 3247, which indicates that a tool is being used. The table below lists the destination hosts:

Possible Red Worm (UDP) traffic from MY.NET.153.35 – Port 65535/udp		
Target	***Resolves To***	***Count***
217.165.75.40	de2294.alshamil.net.ae	13
64.53.153.88	d53-88-153.nap.wideopenwest.com	7
64.231.25.137	HSE-Toronto-ppp293457.sympatico.ca	4
24.5.46.4	c-24-5-46-4.client.comcast.net	3
82.48.87.89	host89-87.pool8248.interbusiness.it	2

Note that destination 24.5.46.4 also had connections to TCP port 65535 from several internal hosts.

Internal systems compromised by worms

Several internal systems were found to perform extensive scanning on TCP ports 135, 139, 445, 1025, 2745, 3127, 3410, 5000 and 6129. Furthermore, those systems contacted one single host on port 7000, triggering the alert "*[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC*". SdBot was first analysed back in 2002, which would explain why the (presumably old) Snort ruleset found it. SdBot was originally[34] nothing more than an IRC backdoor, but later variants appeared with more offensive elements.

The characteristics of this attack says it should be a Phatbot, Agobot, Gaobot, Polybot - or even a recent SdBot - variant. It is not really important which one it is - what matters is that several internal systems are infected by a worm with an IRC-controlled backdoor (that is, controlled by an malicious outsider), and that they currently are scanning external hosts in order to add to a potential zombie army. By having a clue about the malware in question, we can also find a likely explanation what exactly it is looking for[35]. An example on the scan distribution, including targeted vulnerabilities, is shown in the table below.

Scan distribution from MY.NET.112.152			
Port	***Probable Target Service***	***Unique Targets***	***Packets***
2745	Bagle Backdoor	130,315	166,818
135	DCOM2 vulnerability. (MS03-039)	118,213	151,458
1025	MS03-032 vulnerabilities	108,950	138,439
445	MS03-032 vulnerabilities	98,122	124,548
3127	MyDoom.A Backdoor	91,188	115,350
6129	Dameware vulnerability	85,436	107,441
139	MS03-049 Workstation vulnerability	78,095	98,054
3410	Optix Pro Backdoor	73,142	91,474
5000	MS01-059 UPnP vuln.	68,893	85,981

Although the number of packets and targets vary between the source addresses, the pattern is similar (i.e. port 2745 was hit more than port 5000). The observant reader will have noticed that the unique targets are fewer than the packet count. This has an simple explanation: if more than one million hosts are scanned, the likelihood is that some of them will have the targeted port open. When this happens, an attempt to exploit the vulnerability will occur. As TCP is the protocol in use, a three-way-handshake will be required before the malicious payload. Thus, for each port that is open, at least 4 additional packets will be sent.

The below table shows which internal hosts seems to be infected, and the extent of their scanning (destination ports aggregated).

34 http://www.f-secure.com/v-descs/sdbot.shtml [37]
35 Courtesy of Joe Stewart [47]

Probable Worm activity from internal systems			
T10S	***Source***	***Unique Targets***	***Packets***
5	MY.NET.70.96	89,939	1,130,728
6	MY.NET.112.152	139,466	1,079,563
8	MY.NET.66.56	37,160	338,527
10	MY.NET.42.2	39,832	250,968
-	MY.NET.151.75	24,345	211,495
-	MY.NET.80.224	15,466	139,721
-	MY.NET.153.174	16,760	136,783
-	MY.NET.43.10	14,521	81,710
-	MY.NET.80.5	9,582	81,588
-	MY.NET.97.66	17,872	73,547
-	MY.NET.150.199	5,755	47,867

At this point, I also noticed that our old friend MY.NET.84.235 seem to be suffering from a SdBot as well. Not only that, but it goes to the same server as MY.NET.80.5, which may be suffering from yet another type of worm.

3.10. Out Of Spec Traffic

Traffic with "Reserved" TCP Flags

Most of the OOS packets are probably logged due to the fact that the Snort ruleset is a bit outdated. For example, the number one source on the T10O list is responsible for 1716 POP3 (port 110 TCP) attempts towards MY.NET.6.7, which may or may not be a POP server. Apparently, one new session is made every minute through the data set. There are no recorded alerts from the source.

It is fairly clear why the traffic triggered. All packets have the TCP "reserverd" bits set (both 1 and 2). This is the problem with the older ruleset; those bits are reserved no longer, but instead used for Explicit Congestion Notifications[36]. And in the SYN packet, both CWR and ECE (bits 1 and 2) should be set. Of course, for the packet to be completely valid, the ECN bits in the IP header should both be set to zero.

It is possible (and even likely) that MY.NET.6.7 is a POP server, and that 68.54.84.49 is a remote user doing a frequent (a bit too frequent to be polite, in my opinion) connections to retrieve new email in almost real-time. But as nobody told Snort that the reserved bits are no more, the alerts are still being generated.

This issue seem to apply for sources 1 to 8 and 10 on the T10O list. An overview on the traffic is available below.

36 Implemeted by RFC 3168 [48].

Traffic Overview (Reserved TCP Flags)					
T10O	***Source***	***Destination(s)***	***Port(s)***	***Usage***	***Count***
1	68.54.84.49	MY.NET.6.7	110	POP3	1,716
2	202.144.28.167	MY.NET.70.225	4662	File Sharing	699
3	141.224.64.4	MY.NET.12.4	25	SMTP	265
4	62.174.236.17	MY.NET.43.3	24842	<unknown>	171
5	193.170.194.27	MY.NET.110.82 MY.NET.12.6	113 25	Auth/Ident SMTP	169
6	66.225.198.20	MY.NET.12.6	25	SMTP	166
7	80.54.249.136	MY.NET70.225	4662	File Sharing	163
8	80.38.206.68	MY.NET.42.10	28053	<unknown>	116
10	202.54.60.162	MY.NET.24.44 MY.NET.60.14	80	HTTP	90

Nothing strikes my mind as odd when looking at the timestamps either, so it is probably nothing serious (unless file sharing is prohibited, but that is a policy question rather than a OOS problem).

Something that is somewhat out of the ordinary here is the traffic to ports 24842 and 28053, which are not standard ports. The traffic has the SYN flags set, so it is apparently attempts to create new sessions.

Another oddity came from 80.54.249.136: it appears that ephemeral ports are capped at 5000, the largest one seen is 4982, but the traffic indicates several wrap-overs, i.e. it reverts back to somewhere just above 1023.

Traffic from MY.NET.199.202

MY.NET.199.202 has position 9 on T10O, with 96 recorded packets. The reason they were recorded seems to be same as above, i.e. “reserved” TCP flags. However, there is something peculiar with those packets - these are the only ones in scans, oos, or alerts which have MY.NET both as source and destination. One might suspect something unsavoury going on. The TTL is 61 in all packets (the initial TTL may have been 64, which means the host is three router hops away), and the source port is increasing incrementally (as it should), so it is most likely the same source host. It could be that MY.NET.199.202 has a misconfigured router setting, which could explain why the traffic is seen by the Snort sensor.

The traffic can be summarised thus:

OOS Traffic from MY.NET.199.202		
Target	***Port***	***Number of Packets***

OOS Traffic from MY.NET.199.202		
MY.NET.12.7	443	39
MY.NET.24.34	80	29
MY.NET.12.4	143	10
MY.NET.60.39	22	4
MY.NET.60.38	22	4
MY.NET.24.44	80	3
MY.NET.60.14	80	2
MY.NET.24.33	443	2
MY.NET.12.2	25	2
MY.NET.34.11	80	1

Other OOS traffic from MY.NET

All other OOS packets from MY.NET comes from two hosts: MY.NET.12.4 and MY.NET.12.6. An overview is available below:

OOS traffic from MY.NET.12.4			
Target	***Resolves To***	***SRC port***	***DST Port***
68.34.99.7	pcp02751671pcs.essex01.md.comcast.net	143	19830
66.92.148.23	dsl092-148-023.wdc2.dsl.speakeasy.net	143	3941
24.16.14.104	c-24-16-14-104.client.comcast.net	143	2321

From looking at the source port, one might expect this to be IMAP server responses.

OOS traffic from MY.NET.12.6			
Target	***Resolves To***	***SRC Port***	***DST Port***
211.144.32.98	<not available>	25	3207
216.219.110.229	mail16.savings-blvd.com	25	13808
24.10.168.33	c-24-10-168-33.client.comcast.net	25	4303
24.187.74.24	ool-18bb4a18.dyn.optonline.net	25	3389
61.1.93.44	<not available>	25	1236
69.6.65.45	<not available>	25	19022
24.57.150.188	d57-150-188.home.cgocable.net	25	3150
216.219.110.139	mail6.savingsave.com	25	4382

My guess is that MY.NET.12.0/24 is dedicated for email services (we previously saw a system – MY.NET.12.7 - getting traffic on the POP3 port). MY.NET.12.6 seem to be a SMTP server.

Checking the oos.txt file in more depth, the traffic to those two hosts are TCP RST- it fits perfectly well into the theory of it being responses.

There is three interesting things with this traffic:

The TTL is 255

This means that the NIDS sensor should be on the MY.NET.12.0 network segment or that the traffic is crafted by a device on the same segment as the sniffer interface (The ECN specification warns that some routers or firewalls may wrongly respond with a RST when seeing ECN bits set).

The RST packets have the CWR/ECE bits set

This is kind of interesting. The ECN specification is a bit hard to read, but if I understand correctly, a RST that comes as a response to a SYN should *not* have these bits set.

No corresponding requests

If they are responses, one surmise we would have had requests logged in the same way. That is not the case. And as stated above, ECN-capable (and compliant) systems should not use these bits unless they appeared in the SYN (and they furthermore responded correctly with SYN-ACK), in which case they too would have been logged. One possibility is that there is multiple routes to the Internet, and that the replies took a different path out. But for the traffic to be legitimate, these must be resets for an already established TCP session, and not something else (e.g. response to a malformed packet).

To conclude this section – I have no real idea what is going on there. Further investigation is recommended.

3.11. Analysis Process

My analysis platform was a laptop system running Gentoo Linux.

The first step was to concatenate all files of the same type into one file. This gives larger files (the scans file became 970 MB), which increases the time and computing resources required for analysis. The advantage is that it is easier to find relationships over multiple days. The files were uncompressed using gunzip before the concatenation took place.

```
$ cat alert.* >> alerts.txt
$ cat scans.* >> scans.txt
$ cat oos_reports* >> oos.txt
```

A fourth file was created by removing all portscans from the alert file, since these were a majority. My thought was to deal with the portscans through the scan logs instead. This step was done by using grep:

```
$ grep -v "spp_portscan" alerts.txt > alerts_no_ps.txt
```

The four files listed below became the input for all subsequent analysis.

Concatenated files		
File name	***File size***	***Contents***
alerts.txt	207M	All generated alerts
alerts_no_ps.txt	8.6M	All alerts that are not portscans
oos.txt	8.1M	All "Out of Spec" alerts
scans.txt	994M	Full list of portscans

After taking a quick glance at the raw data, I quickly realised the foolishness of sifting through them manually. Being somewhat familiar with Perl, it made a logical choice as it is more or less designed for just this sort of things. Or is it?

I noticed that Perl did not like it at all when it ran out of memory (segmentation fault). To be able to do any meaningful analysis I had to add some extra 2 GB of swap and not even then was I able to run all analysis. After having run snortsnarf.pl for over four hours, I gave up and went to install a MySQL database instead.

That caused the next problem. I needed to get the data into the database. And before even that, I had to decide how to design the tables.

I looked at Brandon Newport's practical [49] to get some pointers (for information on how to create the initial database, refer to his practical or the excellent MySQL documentation [50]).

I opted for three different tables (Brandon used two, alerts and spp): alert, scans and oos. After some fiddling on my own part, the three tables looked thus (formatted as from the 'describe' command):

Description of the 'alert' table					
Field	***Type***	***Null***	***Key***	***Default***	***Extra***
date	varchar(21)	YES		NULL	
attack	varchar(80)	YES		NULL	
src	varchar(15)	YES		NULL	
srcport	int(5)	YES		NULL	
dst	varchar(15)	YES		NULL	
dstport	int(5)	YES		NULL	

Description of the 'scans' table					
Field	**Type**	**Null**	**Key**	**Default**	**Extra**
date	varchar(15)	YES		NULL	
src	varchar(15)	YES		NULL	
srcport	int(5)	YES		NULL	
dst	varchar(15)	YES		NULL	
dstport	int(5)	YES		NULL	
type	varchar(10)	YES		NULL	
info	varchar(30)	YES		NULL	

Description of the 'oos' table					
Field	**Type**	**Null**	**Key**	**Default**	**Extra**
date	varchar(25)	YES		NULL	
src	varchar(15)	YES		NULL	
srcport	int(5)	YES		NULL	
dst	varchar(15)	YES		NULL	
dstport	int(5)	YES		NULL	

For the amusement of the reader, here is a “what ever did I think with”-story:

> Some time into the analysis, I tried to list all destination ports below 1024 and got back entries with port 65535. My error?
>
> The “dstport” field was initially defined as “varchar” instead of “int”. Text and numerical comparison behaves quite differently. Needless to say, that was the point where I had to learn how to use "alter table".

I decided to go without the portscanning alerts already extracted from the alert file since I would be analysing each individual packet as part of the scans data instead.

Preparing the data and populating the tables

When loading data into a database, the data must be formatted in a way the database can understand. Usually, this means an arbitrary number of lines, separated by newlines, and consisting of a number of fields, delimited by a character that is unlikely to appear in the data. Brandon used % (not perhaps the most common choice) as delimiter, but I saw no particular reason to do otherwise.

Furthermore, I extended the sed lines he wrote to do the extraction, and combined them with grep. For more information what the particular lines does, please refer to his practical (my sed modifications include extra white space removal; the grep command is used to exclude incorrect lines).

Preparing the alert data

As Snort handles several IP-protocols, it should not come as a surprise that some alerts (e.g. any that is not TCP or UDP) lacked port number information. Obviously, this wreaked some havoc in the import process. Result: a small program written (I decided I would prefer to have all alerts in the same table rather than splitting them into two). The program checks the number of delimiters, and if it is three (as opposed to five, as it will be with port numbers), it inserts two more delimiters in the correct position (which will cause the port values to be null). Rows that has neither three nor five delimiters will be excluded.

```
$ grep "^04/" alerts_no_ps.txt \
      | sed -e 's/[[:space:]]*\[\*\*\][[:space:]]*/\%/g' \
      -e 's/ -> /\%/g' -e 's/:/%/4' -e 's/:/%/3' \
      | ./insert_delims
```

A file named file db_alert_input.txt will be generated by insert_delims (which is the program I wrote[37]). 15 rows were ignored by insert_delims, and 79 were excluded by grep. They were reviewed to confirm they did not contain any critical alerts.

Two rows had to be corrected manually (the \% in K:lined were replaced with a dash), as the sed script was unable to handle colons in the alert description (I preferred this manual correction rather than spending a night in sed's manual page):

> 04/07-19:53:19.847607%[UMBC NIDS IRC Alert] K\%line'd user detected, possible trojan.%211.146.117.228%6883%MY.NET.84.203:1181
>
> 04/10-02:26:55.616670%[UMBC NIDS IRC Alert] K\%line'd user detected, possible trojan.%210.155.158.200%6969%MY.NET.97.158:3416

After this, the data loaded without problems:

```
mysql> load data local infile "/home/gcia/db_alert_input.txt"
      into table alert fields terminated by "%";
```

Preparing the scans data

When I first tried to import the data into the tables, MySQL returned warnings. A cursory glance showed incomplete lines in the downloaded files. I tried to see if the original meaning could be preserved, but in many cases, the lines were too corrupted. I decided to write a small program to remove most of the offending lines (the program[38] ignores all lines which does not have exactly *n* delimiters and the output is written to dbout.txt).

```
cat scans.txt | sed -e 's/Apr  /Apr /' -e 's/ -> /%/' \
      -e 's/:/%/3' -e 's/:/%/3' -e 's/ /%/3' \
      -e 's/ /%/3' -e 's/ /%/3' \
      | ./extract_dbout 6
```

37 It can be retrieved from http://hem.bredband.net/flank/source/insert_delims.c [51]
38 Available at http://hem.bredband.net/flank/source/extract_dbout.c [52]

This excluded 116 bad rows, and I decided to load the data:

```
mysql> load data local infile "/home/gcia/dbout.txt"
       into table scans fields terminated by "%";
```

Unfortunately, there were still warnings during import. To remove remaining bad lines, I chose to do it from MySQL instead (remember, there were more than 15 million entries in there so it was not very funny to handle it with standard UNIX commands, and especially not on laptops which usually comes with IDE disks).

The following command listed all scan types in the table:

```
mysql> select distinct type from scans;
```

I could then list all scan types which looked legitimate, and tell MySQL to delete any other types. This left me with a table I felt were OK for my need. Probably not the most elegant way to do it, but it worked for me.

```
mysql> delete from scans where type != 'SYNFIN' and
       type != 'SPAU' and type != 'FULLXMAS' and type != 'XMAS'
       and type != 'NMAPID' and type != 'NOACK' and
       type != 'NULL' and type != 'UNKNOWN' and
       type != 'INVALIDACK' and type != 'SYN' and type != 'FIN'
       and type != 'UDP' and type != 'VECNA';
```

Preparing the OOS data

As I already mentioned, I decided to import the OOS information into the database. For this purpose, I used only the first line for each alert to make it manageable (it contains time, source, destination and ports). The oos table was used to find relationships, while the text file was used for in-depth analysis on isolated events.

```
$ grep "^04/" oos.txt | sed -e 's/ -> /%/' -e 's/ /%/' \
       -e 's/:/%/3' -e 's/:/%/3' > oos_db_in.txt

mysql> load data local infile"/home/gcia/oos_db_in.txt"
       into table oos fields terminated by "%";
```

Performing the analysis

With the database up and running, it was time to start the analysis. It was done with a large number of different SQL queries. It is not feasible to describe every command used, or even a small subset of them. Instead, I will just give a few samples to indicate my methodology.

I thought it a good idea to start with the Top Ten Talkers part, and then delve deeper into interesting events.

For example, to list the top ten generators of alerts, the following SQL query could be used ("desc" means it should list the highest count first):

```
mysql> select src, count(*) as count from alert group by src
       order by count desc limit 10;
```

Then, assuming that MY.NET.11.7 was highest on the list, one could dig further into its behavior (this would list all destinations, the alert, and the number of occurrences for each destination and alert, with the highest count first):

```
mysql> select dst, alert, count(*) as count from alert
       where src = 'MY.NET.11.7' group by dst,alert
       order by count desc;
```

The great advantage with the database approach is its flexibility. The problem is that is hard to tell exactly what to look after. That will vary between each data set. As such, I felt it more important to cover how I prepared the data, which I hope I have done to a satisfying degree.

Since the Snort ruleset which generated the logs was a bit dated, I downloaded an older version of Snort to use as a reference. For this purpose, I chose version 1.8.5.

I also want give some extra credit to http://www.portsdb.org [53], which is an excellent resource for finding information regarding port usage (both IANA [54][55] and user supplied).

Method Evaluation

I am not a DBA, and my SQL and database management skills are rudimentary at best (the only prior experience I had was the odd SQL query now and then). This being the case, I am sure that my queries were coarse, and that I could have designed the tables and database much more optimally. Even so, I felt many times during the analysis that I had done the correct choice. The worst query took approximately 4 minutes (it was done on the 15 million entries in the scans table). To just write a perl script for that particular query would have taken far longer, let alone the execution of it.

The biggest problem was to get the data into the tables, and this was further complicated by the invalid entries that existed in the data files.

Something that I realised at the end of the analysis (I never fixed this) was that I should have taken more care with the timestamps. If they had been imported into fields of date/timestamp type, it could have simplified some searches (e.g. being able to sort on reverse date, or event within an hour). I feel I managed pretty well without this, but I thought I should mention it as a suggestion for future work.

4. References for part two and three

[0] ”Snort”. URL: http://www.snort.org/ (25 Jun 2004)

[1] “Part 2 data” URL: http://www.incidents.org/logs/Raw/2003.12.15.tgz (24 Jun 2004)

[2] Martin, Ian. “SANS GCIA Practical Version 3.3”. Jul 2003.
URL: http://www.giac.org/practical/GCIA/Ian_Martin_GCIA.pdf (24 Jun 2004)

[3] “oui.txt”. URL: http://standards.ieee.org/regauth/oui/oui.txt (24 Jun 2004)

[4] “IEEE OUI and Company_id Assignments”
URL: http://standards.ieee.org/regauth/oui/index.shtml (24 Jun 2004)

[5] ”GCIA Raw data README”. Apr. 2004.
URL: http://www.incidents.org/logs/Raw/README (24 Jun 2004)

[6] Postel, Jon (editor). ”Internet Protocol”. Sep 1981.
URL: http://www.ietf.org/rfc/rfc0791.txt (24 Jun 2004)

[7] “Nessus”. URL: http://www.nessus.org (24 Jun 2004)

[8] “Source routed packets”. 2003.
URL: http://cgi.nessus.org/plugins/dump.php3?id=11834 (24 Jun 2004)

[9] “CAN-1999-0510”. 1999.
URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0510 (24 Jun 2004)

[10] “Source Routing”. URL:
http://www.iss.net/security_center/advice/Underground (line wrapped)
/Hacking/Methods/Technical/Source_Routing/default.htm (24 Jun 2004)

[11] Novak, Judy et al. “TCP/IP for Intrusion Detection”. The SANS Institute. 2004.

[12] “IDS418 "SOURCE_ROUTE_UDP_LSRR"”.
URL: http://www.whitehats.com/info/IDS418 (24 Jun 2004)

[13] “CVE-1999-0909”. 2000.
URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0909 (24 Jun 2004)

[14] “Microsoft Windows IP Source Routing Vulnerability”. Sep 1999.
URL: http://www.securityfocus.com/bid/646 (24 Jun 2004)

[15] “NAI-Sep201999: Windows IP Source Routing Vulnerability”. Sep 1999.
URL: http://www.securityfocus.com/advisories/1761 (24 Jun 2004)

[16] Postel, Jon (editor). ”Transmission Control Protocol”. Sep 1981.
URL: http://www.ietf.org/rfc/rfc0793.txt (24 Jun 2004)

[17] ”OpenSSH Security”. URL: http://www.openssh.org/security.html (25 Jun 2004)

[18] Cheswick, William & Bellovin, Steven. “Firewalls and Internet Security Repelling the Wily Hacker”. Reading. Addison-Wesley. 1994. ISBN 0-201-63357-4.

[19] Sternudd, Patrik. “[Intrusions] LOGS: GIAC GCIA Version 3.4 Practical Detect Patrik Sternudd”. Apr 2004. URL: http://www.dshield.org/pipermail/intrusions/2004-April/007902.php (24 Jun 2004)

[20] Sternudd, Patrik. “[Intrusions] LOGS: GIAC GCIA Version 3.4 Practical Detect Patrik Sternudd (resubmitted)”. Apr 2004.
URL: http://www.dshield.org/pipermail/intrusions/2004-April/007945.php (24 Jun 2004)

[21] “Unicode 4.0.1”. Mar 2004. URL: http://www.unicode.org/versions/Unicode4.0.1/ (24 Jun 2004)

[22] “Windows Workstation Service Remote Buffer Overflow”. Nov 2003.
URL: http://www.eeye.com/html/Research/Advisories/AD20031111.html (24 Jun 2004)

[23] “Microsoft Security Bulletin MS03-049”. Nov 2003. URL:
http://www.microsoft.com/technet/security/bulletin/MS03-049.mspx (24 Jun 2004)

[24] “CAN-2003-0812”. 2003.
URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0812 (24 Jun 2004)

[25] “W32.HLLW.Deadhat.B”. Feb 2004. URL:
http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.deadhat.b.html (24 Jun 2004)

[26] ”Nmap”. URL: http://www.insecure.org/nmap/ (24 Jun 2004)

[27] “W32.Mydoom.A@mm”. Jan 2004. URL:
http://securityresponse.symantec.com/avcenter/venc/data/w32.novarg.a@mm.html
(24 Jun 2004)

[28] “Deadhat.B”. Feb. 2004. URL: http://www.pandasoftware.com/virus_info (line wrapped)
/encyclopedia/overview.aspx?lst=det&idvirus=44590 (24 Jun 2004)

[29] Conelly, Ken. “[LOGS] Summary of large-scale portscanning detects”. Feb 2004.
URL: http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00105.html (24 Jun 2004)

[30] Schwartzkopff, Michael. “Source port 22002”. Feb 2004.
URL: http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00126.html (24 Jun 2004)

[31] Gibbons, Carl. “Re: [LOGS] Summary of large-scale portscanning detects”. Mar 2004.
URL: http://cert.uni-stuttgart.de/archive/intrusions/2004/03/msg00032.html (24 Jun 2004)

[32] Sternudd, Patrik. “[Intrusions] Source port 22002 (Worm targeting MyDoom backdoors)”.
Apr 2004. URL: http://www.dshield.org/pipermail/intrusions/2004-April/007888.php (24 Jun 2004)

[33] Rekhter, et al. ”Address Allocation for Private Internets”. Feb 1996.
URL: http://www.ietf.org/rfc/rfc1918.txt (23 Jun 2004)

[34] IANA. “Special-Use IPv4 Addresses”. Sep 2002.
URL: http://www.ietf.org/rfc/rfc3330.txt (23 Jun 2004)

[35] eDonkey2000. URL: http://www.edonkey2000.com/ (24 Jun 2004)

[36] Rautiainen, Sami. “F-Secure Virus Descriptions : Adore”. Apr 2001.
URL: http://www.f-secure.com/v-descs/adore.shtml (24 Jun 2004)

[37] “F-Secure Virus Descriptions : SdBot”. Nov 2002.
URL: http://www.f-secure.com/v-descs/sdbot.shtml (24 Jun 2004)

[38] “GIAC Logs”. URL: http://isc.sans.org/logs/ (24 Jun 2004)

[39] “IDS177 "NETBIOS-NAME-QUERY”.
URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids177&view=research (24 Jun 2004)

[40] Beardsley, Tod. “Intrusion Detection and Analysis: Theory, Techniques, and Tools”. May 2002.
URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.pdf (23 Jun 2004)

[41] “eDonkey2000 FAQ”.
URL: http://www.edonkey2000.com/documentation/donkeyfaq.html (24 Jun 2004)

[42] “eDonkey2000 FAQ – General”.
URL: http://www.edonkey2000.com/documentation/clientfaq.html (24 Jun 2004)

[43] “Novell Protocols”. URL: http://www.protocols.com/pbook/novel.htm#NCP (24 Jun 2004)

[44] “Overview of NetStorage”. Jun 2002.
URL: http://developer.novell.com/research/appnotes/2002/june/03/a0206033.htm (24 Jun 2004)

[45] Gordon, Les. “Intrusion Analysis - The Director's Cut!”. Nov 2002.
URL: http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc (24 Jun 2004)

[46] “broadband » Forums » Stopping Spam » Vast amount of spam linked to one "company"”.
May 2004. URL: http://www.dslreports.com/forum/remark,10137492~mode=flat (24 Jun 2004)

[47] “Security Incidents: Re: Outbreak of a virus on campus, scanning tcp 80/6129/1025/3127”.
Apr 2004. URL: http://seclists.org/lists/incidents/2004/Apr/0063.html (24 Jun 2004)

[48] Ramakrishnan, et al. ”The Addition of ECN to IP”. Sep 2001.
URL: http://www.ietf.org/rfc/rfc3168.txt (23 Jun 2004)

[49] Newport, Brandon. “Level Two Intrusion Detection In Depth GCIA Practical Assignment”.
May 2001. URL: http://www.giac.org/practical/Brandon_Newport_GCIA.zip (23 Jun 2004)

[50] ”MySQL Reference Manual”. URL: http://dev.mysql.com/doc/mysql/en/index.html (25 Jun 2004)

[51] Sternudd, Patrik. “insert_delims.c”. Jun 2004.
URL: http://hem.bredband.net/flank/source/insert_delims.c (24 Jun 2004)

[52] Sternudd, Patrik. “extract_dbout.c”. Jun 2004.
URL: http://hem.bredband.net/flank/source/extract_dbout.c (24 Jun 2004)

[53] ”Ports Database”. URL: http://www.portsdb.org/ (24 Jun 2004)

[54] Reynolds, Joyce K (editor). ”Assigned Numbers: RFC 1700 is replaced by an On-Line Database”.
Jan 2002. URL: http://www.ietf.org/rfc/rfc3232.txt (25 Jun 2004)

[55] IANA. ”TCP and UDP Port Numbers” 19 Mar 2004.
URL: http://www.iana.org/assignments/port-numbers (25 Jun 2004)