

## Windows Forensics - A Case Study: Part Two

Stephen Barish 2003-03-06

This article is the second in a two-part series that will offer a case study of forensics in a Windows environment. In [Part One](#), we discussed host-based forensics techniques that first responders can use to detect attacks in relatively unprotected environments, and how to begin collecting information to determine the appropriate response. Part One dealt with understanding what an attacker was doing on an individual host. This article deals with determining the scope of the compromise, and understanding what the attacker is trying to accomplish at the network level. Along the way, we'll be discussing some tools and techniques that are useful in this type of detective work.

### The Objective

As we left off last time, we had discovered the compromise (although we have yet to identify the compromise method), identified the post-attack "fingerprint" of this particular group, and begun to understand what was happening in the enterprise. Before we start with the eradication phase of our incident response, we really need to complete the identification phase: we have yet to identify the initial compromise method, or to identify the scope of the compromise!

At this point in the investigation, we have reason to believe the attackers are making illicit use of the victim network to serve content to their friends and neighbors. While it is possible any individual content provider might not mind serving some of this material (the kind that isn't unlawful anyway), our victim network isn't getting paid for this service, and the attackers have a free ride. Of more concern, the investigation so far has yielded information that indicates the attackers have compromised both local and domain Administrator accounts in our enterprise.

Our objectives moving forward are simple. We want to determine how widespread the attacker's control over the network is, what the initial compromise method was, and who the attacker is (if possible).

### Developing a Toolkit

It would be great if all of us had fully monitored environments and the staff required to correlate and analyze all the security data that can be extracted from such an enterprise. Unfortunately, the reality of the situation is that most of us do not have host and network IDS systems, tiered firewalls, strong authentication and authorization systems, and logging is generally insufficient. The problem gets worse in larger environments where the sheer amount of data and systems we're responsible for makes it extremely difficult to perform even "near-real time" analysis. So when the inevitable incident occurs, we need a "quick and dirty" toolkit and methodology to assess what is happening on the network, so we can make smart decisions on remediation measures.

I typically like to use a laptop loaded with both Windows 2000 (and the Windows 2000 Resource Kit) and Linux to conduct my network assessments, and a variety of tools that operate in both environments.

Tool	Windows	Unix
Network Sniffer(s)	Windump, Ethereal	Tcpdump, Ethereal, dsniiff

IDS Software	Snort	Snort
Analysis Software		EtherApe, tcpreplay
Port Scanners	Fscan, nmapwin	Nmap

These are just a few tools that are available, and represent my personal favorites - your mileage may vary. However, by using the capabilities these tools and others like them offer, we can usually develop a good picture of what is happening on the network, and what an attacker is doing.

## Getting Started

Before we even consider launching a sniffer, we need to determine what data we're looking for in the network traffic. In a 100Mbps switched network, it's trivial to overwhelm a sniffer, an analysis workstation, or an analyst. I like to start by looking at a victim box, preferably one I know is still being used by the attacker. In a switched environment, which is really the worst case from a traffic analyst's perspective, I like to set up on a mirror port tap into the network connection between the victim box and the switch. Provided the switch supports full duplex mirroring of an individual port, it is highly preferable to monitor from the switch rather than interrupt network connectivity, as it introduces much less of a signature that the attacker might notice. While it is possible to insert a sniffer between the switch and the monitored host using an Ethernet tap, it's a distinct second choice. Given a choice, I almost always prefer to set up on a mirror port.

The advantage to setting up on a mirror port rather than using an Ethernet tap is that you can quickly reconfigure the switch to allow monitoring of another individual host or VLAN (provided the switch will support the desired configuration). Although mirroring a highly utilized VLAN is almost sure to cause packet loss and collision problems on some switches, I've found the technique can help get a "quick and dirty" view of what's going on. It's certainly not the most desirable solution from a long-term IDS perspective, but it will help us monitor and analyze incoming traffic.

## Windows Native Tools

For those of who grew up sniffing traffic in a largely Unix environment, there is good news! Most of the tools we are used to now exist in some form or other in the Windows environment. For instance, the standard [tcpdump utility](#) used in most Unix distributions has been ported with the name [Windump](#). Windump makes use of a port of libpcap (unsurprisingly called [WinPcap](#) to interact with the various network interfaces on a sniffing host virtually identically with [tcpdump](#). Windump reads and writes the binary output of tcpdump, so data can be collected and analyzed in either environment.

After downloading and installing Windump and setting up on the switch's mirror port, we can start capturing and pre-processing traffic. The real power in using Windump is that it allows users to filter traffic prior to recording it in either text, ASCII, or hex formats. By carefully crafting filters, we can minimize the amount of unnecessary traffic collected. For instance, the following command with imbedded filter will find traffic between **victim** and **target**, and will record it to a file.

```
C:\> Windump -il host victim and target -w output
```

In reality, we rarely know what we'll be looking for during the initial data capture, so it is wisest to conduct

full content monitoring. In this case, we want to capture the entire packet, including any payload that may be of interest. Since the Maximum Transfer Unit of Ethernet is 1500 bytes, we set our *snaplength* to 1500 bytes also using the *-s* switch. Finally, we use *-n* to prevent Windump from converting host addresses and port numbers. This will prevent us from sending unintended packets to a DNS server that may be owned by the attacker.

```
C:\> Windump -il -s1500 -n -w output
```

To analyze this data, we could use Windump to read *output* and pass it through a series of filters, but it is much more intuitive to dump the resulting data into a more graphical analysis tool. My personal favorite is *Ethereal*. *Ethereal* provides a graphical interface for analysts to point-and-click their way through the packet, its headers, and down into the data itself. I also like its ability to decode certain protocols and reassemble TCP streams.

The screenshot displays the *The Ethernet Network Analyzer* (Ethereal) interface. The main window shows a list of captured packets with columns for No., Time, Source, Destination, Protocol, and Info. The selected packet (No. 1) is expanded to show its details: Ethernet II, Internet Protocol, and Transmission Control Protocol.

A summary window titled "Ethereal: Ca..." is overlaid on the main window, showing the following statistics:

Captured Frames		
Total	219	(100.0%)
SCTP	0	(0.0%)
TCP	213	(97.3%)
UDP	6	(2.7%)
ICMP	0	(0.0%)
ARP	0	(0.0%)
OSPF	0	(0.0%)
GRE	0	(0.0%)
NetBIOS	0	(0.0%)
IPX	0	(0.0%)
VINES	0	(0.0%)
Other	0	(0.0%)

The summary window also indicates "Running 00:00:34" and includes a "Stop" button.

*Ethereal* has its own extremely powerful filter language, and because it automates a significant amount of the analysis, is often easier to use than *Windump*. Regardless, either tool can be used to collect or conduct extremely sophisticated network analysis - if you have a bit of time on your side.

## What to Look For?

Just as when we conduct a host-based forensic analysis, we need to have a plan for what to investigate. While each incident will have some unique aspects, there are certain general techniques that are common to almost all network analysis. These are:

- Top Talkers
- Top Recipients
- Most Common Ports/Protocols
- Comparison with Known/Valid Traffic
- Coordination / Correlation with Forensic Examination

The first four elements in this list rarely surprise anyone, and in the case study we're investigating, would undoubtedly show the majority of the activity. After all, if policy says NetBIOS should not exist on the management VLAN, its mere presence is a sign of concern (see [Part One](#) of this series). However, coordinating and correlating network analysis with the results of the forensics exercise is often overlooked. Recall from Part One that we found a file on the compromised server that, when executed, attempted to make connections to multiple hosts on the network as the Administrator. By conducting full content monitoring of these hosts and using Ethereal for analysis, we can begin to search for additional anomalies.

```
C:> Windump -i2 host TGT1 -s1500 -w output
```

After collecting data for a while (typically we collect one-hour snapshots that overlap by approximately one minute), we can begin analyzing the resulting data. After loading `output` into Ethereal, we construct a filter to look for SMB packets (the protocol underlying Windows File Sharing) and find a host of sessions to and from TGT1, all on the management interface (Interface 2 as reported by Windump -D). As we inspect the traffic, we can see a series of attempts to authenticate to TGT1.

No.	Time	Source	Destination	Protocol	Info
27	4.7277	winbox.victim.com	TGT1.victim.com	SMB	TREE CONNECT ANDX REQUEST, NTLMSSP AUTH
35	5.2552	winbox.victim.com	TGT1.victim.com	SMB	TREE CONNECT ANDX REQUEST, NTLMSSP AUTH
42	6.3521	winbox.victim.com	TGT1.victim.com	SMB	TREE CONNECT ANDX REQUEST, NTLMSSP AUTH

These appear to be requests for authentication to a Windows file share. The fact that we are seeing rapid connections from **winbox.victim.com** to **TGT1.victim.com** is somewhat suspicious also, as it represents some automated process running the connection attempts. By further filtering with Ethereal, and using its native protocol decoding capabilities, we can reconstruct the following sequences:

```
FILTER: (ip.addr eq 192.168.254.2 and ip.addr eq 192.168.254.205) and (tcp.port eq 1095 and tcp.port eq 445)
```

Resulting Stream (Excerpt1)

```

00000389  00 00 01 48 ff 53 4d 42 73 00 00 00 00 18 07 c8 ...H.SMB s.....
00000399  00 00 00 00 00 00 00 00 00 00 00 00 00 ff fe .....
000003A9  01 08 50 00 0c ff 00 48 01 04 11 0a 00 01 00 00 ..P....H .....
000003B9  00 00 00 a6 00 00 00 00 00 d4 00 00 a0 0d 01 4e .....N
000003C9  54 4c 4d 53 53 50 00 03 00 00 00 18 00 18 00 66 TLMSSP.. .....f
000003D9  00 00 00 18 00 18 00 7e 00 00 00 0c 00 0c 00 40 .....~ .....@
000003E9  00 00 00 0e 00 0e 00 4c 00 00 00 0c 00 0c 00 5a .....L .....Z
000003F9  00 00 00 10 00 10 00 96 00 00 00 15 82 88 e0 4c .....W
00000409  00 41 00 50 00 54 00 4f 00 50 00 73 00 62 00 61 .I.N.B.O .X.A.d.m
00000419  00 72 00 69 00 73 00 68 00 4c 00 41 00 50 00 54 .i.n.i.s .t.r.a.t
00000429  00 4f 00 50 00 a0 22 69 06 b4 2d 12 7f 00 00 00 .o.r.."i ..-....
00000439  00 00 00 00 00 00 00 00 00 00 00 00 00 8b e9 d5 .....
00000449  b8 26 a1 f2 01 06 6b 6c e3 62 0d 7f fa 63 15 7f .&....kl .b. .c.
00000459  7d d6 64 30 5e d0 ca 7d 5f 30 5f 13 a4 a7 c3 15 }.d0^..} _0_.....
00000469  d1 fb 87 33 8b 00 57 00 69 00 6e 00 64 00 6f 00 ...3..W. i.n.d.o.
00000479  77 00 73 00 20 00 32 00 30 00 30 00 32 00 20 00 w.s. .2. 0.0.2. .
00000489  32 00 36 00 30 00 30 00 20 00 53 00 65 00 72 00 2.6.0.0. .S.e.r.
00000499  76 00 69 00 63 00 65 00 20 00 50 00 61 00 63 00 v.i.c.e. .P.a.c.
000004A9  6b 00 20 00 31 00 00 00 57 00 69 00 6e 00 64 00 k. .1...W.i.n.d.
000004B9  6f 00 77 00 73 00 20 00 32 00 30 00 30 00 32 00 o.w.s. . 2.0.0.2.
000004C9  20 00 35 00 2e 00 31 00 00 00 00 00 .....5...1. ....

```

When Ethereal decodes the SMB datastream, it inserts a "." between characters because most low-level logging and communications in Windows uses Unicode characters. In Unicode, US characters have "00" in the high order byte. Ethereal decodes all non-ASCII characters as a "." The character stream "W.I.N.B.O.X.A.d.m.i.n.i.s.t.r.a.t.o.r." indicates the Administrator of **winbox** is attempting to connect to the target (TGT1).

As expected, the Administrator is attempting to authenticate to TGT1. However, by scanning through the other packets of this type, we uncovered several other usernames including INET\_GLOBAL, a shared account for configuring Web servers, and HELPTECH1, the username of a Help Desk employee, and USERCHUCK who was a high-level programmer in the company. While it would be nice to jump to conclusions and simply grill USERCHUCK or HELPTECH1, we really have no proof of any wrongdoing by either user at this point.

Ethereal is particularly useful when conducting analysis of this sort, as you can page rapidly through the hex representation of the data packet (as shown above) and look for anomalies. However, in a real-world response scenario, it can take hours to start breaking out usernames and passwords this way thanks to the density of data on the network. Although in some cases this type of analysis is the only way to actually break back communications between compromised hosts and the attackers, there are some tools that make life even easier on the analyst. Mike Davis has ported Dug Song's popular [dsniff](http://www.datanerds.net/~mike/) package to Windows ( <http://www.datanerds.net/~mike/>), which will decode usernames and passwords for a whole host of Internet protocols including FTP, Telnet, HTTP, POP, NNTP, IMAP, SNMP, LDAP, Rlogin, NFS, SOCKS, X11, IRC, AIM, CVS, ICQ, Napster, Citrix ICA, Symantec pcAnywhere, NAI Sniffer, Microsoft SMB, and Oracle SQL.

## Looking for Intrusions

By now it should be fairly obvious what a time-consuming process analyzing network traffic can be when using

simple tools like Windump. Although Ethereal provides more useful information to the analyst, it really doesn't *detect* problems; rather, it just provides us with strong analysis tools to investigate the traffic we've collected. When we expand our traffic analysis from the individual host to the network at large, the problems of data saturation expand immensely. To counter this, we need employ some more sophisticated tools.

**Snort**, a packet capture engine and lightweight IDS written by Martin Roesch is extremely useful for this type of analysis. Snort 1.9 binaries are available for the Win32 platform, including the ability to log to a MySQL or MSSQL database. With the addition of a graphical front-end such as **ACID** I can develop statistical analyses automatically (top talkers, most common protocols, etc.) without having to investigate each packet manually. Additionally, by deploying custom signatures and alerts, Snort can greatly reduce the analysis burden. For instance, the rules below will detect any NetBIOS connections to drives F, G, H, and I on target systems (as we saw in the batch file in [Part One](#)).

```
alert tcp any -> $HOME_NET 139 (msg:"NETBIOS SMB F$access"; flow:to_server,established; content:"\
\F$|00 41 3a 00|";)
```

```
alert tcp any -> $HOME_NET 139 (msg:"NETBIOS SMB G$access"; flow:to_server,established; content:"\
\G$|00 41 3a 00|";)
```

```
alert tcp any -> $HOME_NET 139 (msg:"NETBIOS SMB H$access"; flow:to_server,established; content:"\
\H$|00 41 3a 00|";)
```

```
alert tcp any -> $HOME_NET 139 (msg:"NETBIOS SMB I$access"; flow:to_server,established; content:"\
\I$|00 41 3a 00|";)
```

These rules will trigger alerts if any host attempts to mount network drives on the network address assigned to the variable HOME\_NET. By changing the value of HOME\_NET as we change the traffic flowing the mirror port, we can slowly expand the dimensions of our search. These signatures will ultimately allow us to monitor the entire Management VLAN for attempted NetBIOS sessions. By modifying the rules or adding additional rules to look for more network resources, drives, etc., we can begin to build a picture of how large the compromise is. Since we know that by policy no NetBIOS can be present on this VLAN (or any other unencrypted traffic), any alerts represent probable attacker activity.

When I start monitoring network traffic this way, I almost always run two simultaneous captures. By running Snort in real-time with a tailored signature set, I can monitor for ongoing activity and attacks (assuming I've been able to characterize the attacker's behavior and develop custom signatures). But I need to continue full-content monitoring in parallel in order to assure I can conduct more detailed off-line analysis if required. This can be extremely important as in most investigations we discover more about the attacker's methods and behavior the longer the investigation goes on. By maintaining the original data, we can re-run it through Snort with an updated rule set whenever we desire by simply running

```
C:\SNORT> snort -r Windumpfile -c C:\SNORT\snort.conf
```

Remember from Part One of this series that NetBIOS is not permitted on the Management VLAN. Therefore, any NetBIOS access represents suspicious traffic at a minimum, and may range from misuse to abuse or compromise. We can use Ethereal to search `windumpfile` for the IP addresses that Snort identifies and

look manually for usernames that may be compromised.

## Wrapping It Up

The network side of the analysis is often much more time consuming than the "live approach" of a compromised host. As analysts, we have to learn to discriminate between "normal" traffic and the traffic we believe to be abnormal and/or possibly malicious. In the first part of this series, we said that we have the home-field advantage over our attackers. After reading the numerous emails I've received from the field, I'd say that it may be more accurate to say that "we *ought* to have the home-field advantage." We ought to know the policies our networks operate under, and those policies ought to be complied with. While this is not always the case in reality, it is certainly something we ought to strive for.

Historically the Windows environment has been difficult to monitor, analyze, and secure, simply because of the lack of security tools that run on the platform. Thanks to [Sysinternals](#) who distribute the ps utilities demonstrated in Part One, companies like Foundstone and @Stake, and the open source world of WinPcap, Windump, Ethereal, Snort, etc. we are now gaining an increasing ability to conduct the type of investigations from Windows that used to only be possible in the Unix environment.

The astute reader will note this series hinted at a possible source of the original compromise, but did not conclusively prove it. The reality of the situation is that this sometimes happens, particularly in complex environments where there is poor access control and monitoring. There are some data sources that were not investigated in this series, such as the Windows Event and Security Logs, Web server logs, etc., and in a real incident these should absolutely be studied. Often there is sufficient data in the enterprise to rule out certain kinds of attacks even if a robust network security monitoring environment isn't in place.

In our case study there were several "cardinal rules" of security that were violated. The batch file in Part One would not have operated successfully as written if the Administrator account was not identical on each host. In several of the cases that inspired this article, the Global Domain Administrator account was actually the same as each Local Administrator account, further reducing the security of the enterprise as a whole. The second major problem was the bad monitoring environment. Clearly some architectural changes and a redesign of the network are called for. Using some of the tools discussed in this series, and with minimal cost, it is possible to build a rudimentary IDS capability, although monitoring it will probably take significant manpower.

At the risk of starting a religious discussion, Windows is neither more or less inherently secure than any other platform - with a few arguable exceptions. It is one of the most attacked operating systems on the face of the planet, thanks to its wide proliferation. As a result, its flaws are exploited often and well. If you are running a large Windows environment, my advice to you is to prepare yourself for the inevitable compromise by doing the following:

- Build a Toolkit
  - [Fscan](#) and [Fport](#) from Foundstone
  - [PS Utilities](#) from Sysinternals
  - Trusted versions of Windows utilities for each version of Windows in your environment
  - Dual-boot Windows / Linux Laptop loaded with Winpcap, Windump, Ethereal, and Snort+MySQL and Acid along with Linux variants for the Linux partition
- Use Windump/tcpdump and tcpreplay + EtherApe to characterize the network

- Implement an IDS - Even a Rudimentary One,/span>

Finally, take the time to prepare yourself. This goes beyond characterizing the environment, although that's a good first step. There are some great books out there to get you started in network analysis, notably [Network Intrusion Detection](#) and [Intrusion Signatures and Analysis](#) by Stephen Northcutt and company. [The Anti-Hacker Toolkit](#) and [Incident Response](#) are two other great source texts (as mentioned in Part One). Most importantly, take the time to dig into the tools and practice using them. Learn their features and filter languages in particular, as these will be infinitely useful to you the more you learn.

## Relevant Links

[Windows Forensics: A Case Study, Part One](#)

*Stephen Barish*

[Privacy Statement](#)

Copyright 2006, SecurityFocus