Examining a Public Exploit, Part 1

by Don Parker
last updated August 11, 2004

To many people, the world of computer security and intrusion detection can often be confusing to understand. As an instructor, many of the people who ask me about intrusion detection and packet analysis often ask the same questions, such as the following: What tools do you use? Can you practice and learn this at home? What kind of knowledge does one need to have? These and other questions figure predominantly.

In this article series these questions will be answered and shown to the reader for evaluation in a lab environment. We'll take a publicly available exploit, which you can download and compile at your discretion, and then analyze how it would be seen on your network and evaluated from a security administrator's point of view.

1. Assumptions; setting up the environment

To make sure that everyone understands what this article will use in terms of operating systems, tools, and topology for our lab environment I will list them now. The topology of the lab network that was used to generate this file is as follows:

One lab desktop, which has SuSE Pro 9.1 running on it. This is the machine from which the exploit itself is launched, and the IP address is 192.168.1.100.

One lab desktop which has Windows 2000 Pro on it. For the purpose of this example, this install of Windows 2000 Pro in the lab is a fresh install with no service packs or updates installed to it. The reason for this is that the exploit we are using is the old but potent RPC DCOM vulnerability that has long been patched by Microsoft, but often still exists in the wild. Current and patched machines would not be vulnerable to the exploit used in this article, which makes the study ideal for learning, yet it does not also provide an easy ability for readers to start attacking machines on the Internet. Additionally, it's important to note that there unfortunately still exists many such unpatched computers around the world today -- and anyone who does a fresh install of Windows 2000 and connects it to the Internet would immediately be vulnerable to a plethora of worms and exploits including this one. The IP address for the Windows test machine is 192.168.1.101

There is no firewall installed on the Windows 2000 Pro desktop, as is common in the wild. Note that by default there are quite a few ports open and listening by default on a Windows installation. Connectivity between these two lab machines is accomplished via a simple switch. You do not need to actually send an exploit over the Internet to simulate an attack. Contained within our exploit code are the system calls and the raw socket code through which the exploit would be sent.

2. Required tools

With the topology and system specifics laid out, we will now move on to the tools that you will need to follow along with the analysis. The following tools can be used on either a Windows 32-bit operating system such as NT/2K/XP or your favorite Linux/Unix distribution. As the majority of users today use some form of Windows, the analysis will be performed using Windows XP.

The tools that you will need to download and install are as follows:

Snort for your specific platform, be it Win32 or Linux. For Windows users just download Snort and install it; it will automatically install itself to the root of your system drive, in other words C:\ for most users.

Snortsnarf is cross-platform tool written in PERL that provides HTML output intended for diagnostics; version 0211111.1 was used for this article. Install this similar to the Snort installation.

Tcpdump and libpcap if you are using Linux/Unix, or windump and winpcap if you are using Windows.

PERL version 5.6. 1 or newer, for either Linux or Windows. If you are running Linux you probably already have Perl installed. For those running Windows, you can install ActivePerl version 5.6.1. The Activestate site requires registration but it's otherwise free.

There is one final piece of the puzzle to get snortsnarf to work and those are the following : JulianDay.pm, TimeZone.pm, and ParseDate.pm. You can download the above sitetime modules.

With that done you can now use snortsnarf. To confirm you have it running first navigate to your snortsnarf directory. Windows uers can issue the following command;

```
C:\snortsnarf.pl -usage
```

This will bring up the snortsnarf usage menu and confirm that all went well with the install and the copying of the time modules.

3. Traffic analysis

The traffic file generated will be collected off of the lab desktop, which is acting as the exploited box (IP address 192.168.1.101), and the file size and complexity of the exploit for this example has been kept to a minimum. This serves to makes things relatively simple, as this may very well be your first introduction to log file analysis.

When the exploit is compiled, the binary log format is known as "little endian". I recorded the file in this format for my own use so that the reader can replay this file through Snort, windump or tcpdump as well as through a protocol analyzer like Ethereal. My suggestion to you, however, for your first analysis and for the purpose of this article is to use a lower-level tool such as tcpdump or windump (win32 port of tcpdump) instead of Ethereal.

The usage of a protocol analyzer abstracts you from certain analysis, as Ethereal or any other analyzer will break out all of the packet contents for you. Using tcpdump or windump on the other hand will force you to manually locate the fields you wish to analyze, and then analyze them for yourself. It is the opinion of this author that it is far better to remain current with TCP/IP at a granular level, as the reader will learn a great deal more through using these tools. As they say, it is important to understand what is "normal" so you can recognize the "abnormal" when you see it.

3.1 Using snort with a test exploit

Our first step will be to build our binary and send it through Snort. Snort, as you may or may not know, is an open source intrusion detection system or IDS for short. Despite being open-source, many believe it is equal or better than some commercial IDS offerings.

You will need to download the publicly available RPC DCOM exploit and compile it on a Linux or Unix machine; it is available through many sources included the one linked to, above. The binary is not provided with this article series for legal reasons (SecurityFocus is owned by Symantec Corp).

With our test exploit in hand, we are now ready to "wash" this binary through Snort. Note that it is fine to use the standard Snort ruleset that comes with the installation for our test. We will invoke snort with the following switches:

```
c:\snort\bin\snort.exe -r file -c c:\snort.conf -A full
```

Where 'file' is the binary file we copied using tcpdump/windump which captured the exploit when it was invoked and launched at the lab machine. Using the -c switch just tells Snort where to find the snort.conf file so that snort knows which rules to use. Lastly the '-A full' tells Snort to enable full logging.

If you receive an error saying that you cannot get write access to logging directory, it is simply because there is no log directory in the directory you are in. In this case can either create a log directory in the directory you are presently in or use the '-l' switch to tell snort where to log to.

3.2. Using snortsnarf

With the above command, Snort will output the alert.ids file. This file is what we will feed to snortsnarf, which in turn will generate some very nicely outputted HTML pages. From the directory where you installed snortsnarf, you can now enter the following command:

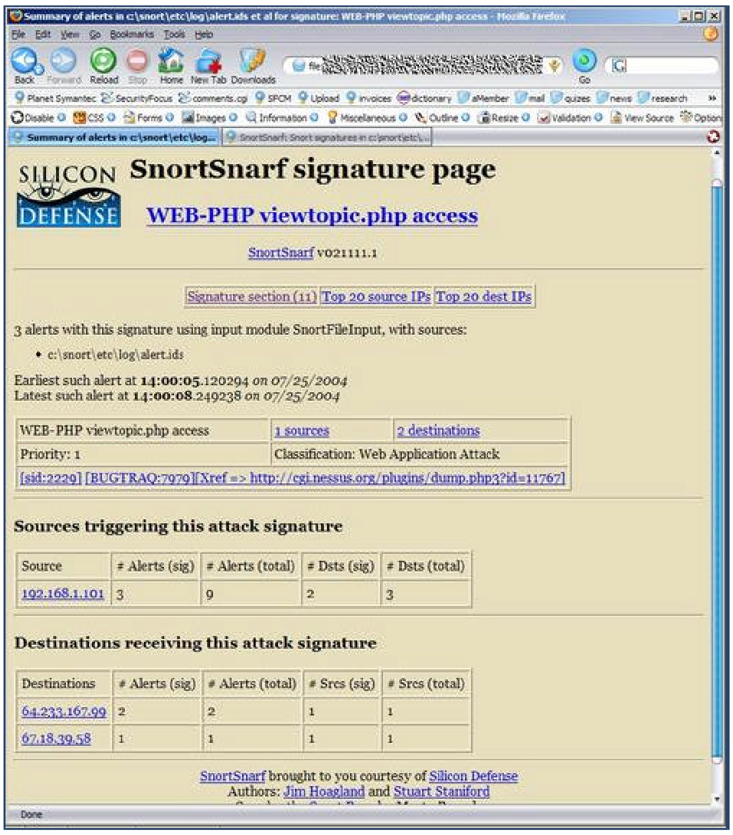
```
snortsnarf.pl alert.ids -win -rs
```

It should take no more then a minute to process this example alert file as it is quite small. Snortsnarf creates a subdirectory called "snfout.alert.ids" which contains all of the HTML output that you will be looking at. At this point you can open the index HTML file locally in your browser. Depending on the version of Snort that you are using you may have more or less then the five alarms that I have, which is as follows:

Priority	Signature (click for sig info)	# Alerts	# Sources	# Dests
1	WEB-PHP viewtopic.php access [sid] [BUGTRAQ]	3	1	2
1	NETBIOS DCERPC ISystemActivator bind attempt [sid] [CVE]	1	1	1
2	TFTP Get [sid]	3	1	1
2	ATTACK-RESPONSES directory listing [sid]	3	1	1
2	ATTACK-RESPONSES 403 Forbidden [sid]	1	1	1

Hyperlinks such to DShield and WHOIS are included as part of the snortsnarf output and are self explanatory, and would have appeared in another column in the above table, so these links have been omitted. This level of details is what makes snortsnarf so nice a tool for use in your have been omitted. This level of details is what makes snortsnarf so nice a tool for use in your analysis, as it includes in one spot many needed resources.

Figure 1, below, shows a screenshot of snortsnarf's detailed output of the first alert.


File Edit View Go Bookmarks Tools Help
Back Forward Reload Stop Home New Tab Downloads Go
Planet Symantec SecurityFocus comments.cgi SPCM Upload invoices dictionary aMember mail quizes news research
Disable CSS Forms Images Information Miscellaneous Outline Resize Validation View Source Options
Done

Summary of alerts in c:\snort\etc\log\alert.ids et al for signature: WEB-PHP viewtopic.php access - Mozilla Firefox

Summary of alerts in c:\snort\etc\log... | SnortSnarf: Snort signatures in c:\snort\etc\...

SILICON DEFENSE

SnortSnarf signature page

WEB-PHP viewtopic.php access

SnortSnarf v021111.1

Signature section (11) | Top 20 source IPs | Top 20 dest IPs

3 alerts with this signature using input module SnortFileInput, with sources:

- c:\snort\etc\log\alert.ids

Earliest such alert at **14:00:05**.120294 *on 07/25/2004*
Latest such alert at **14:00:08**.249238 *on 07/25/2004*

WEB-PHP viewtopic.php access	1 sources	2 destinations
Priority: 1	Classification: Web Application Attack	
[sid:2229] [BUGTRAQ:7979][Xref => http://cgi.nessus.org/plugins/dump.php3?id=11767]		

Sources triggering this attack signature

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)
192.168.1.101	3	9	2	3

Destinations receiving this attack signature

Destinations	# Alerts (sig)	# Alerts (total)	# Srcs (sig)	# Srcs (total)
64.233.167.99	2	2	1	1
67.18.39.58	1	1	1	1

SnortSnarf brought to you courtesy of Silicon Defense
Authors: Jim Hoagland and Stuart Staniford

Figure 1: screenshot of snortsnarf

4. Analyzing Snort alarms

We will now look at the first of the alarms. Number one is the WEB-PHP alarm. You may find the [sid] hyperlinks above useful to read about the alarm itself on the Snort website. Armed with an understanding of the alarm we can now build a bpf filter to isolate the traffic between the two IP addresses for the first instance of this alarm.

Please note as well though that the snortsnarf output also lists, for each specific alarm, the exact packet that caused snort to trigger. Knowing the exact packet to look for is indeed helpful but for the purpose of this article I still prefer to also isolate all traffic between the two hosts involved in a potential intrusion. The bpf filter below is meant to give us only the traffic between the two hosts involved in one of the two instances of this alarm. You will note that I did not use "TCP" as the protocol to pull for, but rather used "IP". Why did I do this? Quite simply because I wanted to make sure I didn't miss any possible precursor activity from either host, such as ICMP packets -- in other words, ICMP ECHO request packets for discovery and other similar packets. If you remember your TCP/IP precepts then you know that all protocols need IP to route them. As a short aside, can you think of an example of a protocol which does not need IP to route it? Think internal broadcasts. ARP does not require IP for routing as it is an internal broadcast protocol and does not actually traverse a router hence the lack of an IP header for routing purposes.

```
C:\ windump.exe -r log_file -nXvSs 0 ip and host 192.168.1.101 and host 64.233.67.99 > web_php
```

What I have done with the above filter is tell windump to parse out all packets with a valid IP header which also has IP addresses 192.168.1.101 and 64.233.67.99 in it, and then output it to a text file called web_php. Note that we will soon see that the 64.x.x.x address in this case is unremarkable, and will simply serve in this example to show a false positive that was generated with normal web surfing and no malicious intent.

One of the switches you may be unfamiliar with is the S switch. This tells windump (or tcpdump) to give you the absolute TCP sequence number versus a relative one. This is a handy switch to use when you want to follow the TCP sequence and Acknowledgement numbers. For instance syn4k.c, the syn flooder, has a hard coded TCP sequence number in it. Having the "S" switch in windump or tcpdump would allow one to see that more clearly.

4.1 Examining packets for the first alert

We will now examine web_php. Though snortsnarf gives us the exact packet to look, at I prefer to take an overall quick look at the packet trace. So I recommend taking a look through the packets until you notice the one noted below.

Seen inside the packet is the probable cause of the alarm. Please note the bolded and underlined portion of the ASCII content. After consulting the alarm description again on the Snort site, it seems like this is a false positive as it does not contain the entire signature, but rather a fragment of it. Also, after further inspection of the packet itself and the remainder of the TCP stream, I see no malicious activity. That being said I will come back to this file later, as in this case I you may not be overly familiar with the alarm. Please note as well that the packet is truncated for I did not want to paste a full packet with a length of 506 bytes into the article:

```
14:00:05.120294 IP (tos 0x0, ttl 128, id 2615, len 506)
192.168.1.101.1050 > 64.233.167.99.80: P [tcp sum ok]
1574142544:1574143010(466) ack 3304341595 win 17393 (DF)
0x0000   4500 01fa 0a37 4000 8006 446d c0a8 0165        E....7@...Dm...e
0x0010   40e9 a763 041a 0050 5dd3 8250 c4f4 405b        @..c...P]..P..@[
0x0020   5018 43f1 dabb 0000 4745 5420 2f75 726c        P.C.....GET./url
0x0030   3f73 613d 5426 6374 3d72 6573 2663 643d        ?sa=T&ct=res&cd=
0x0040   3326 7572 6c3d 6874 7470 2533 412f 2f77        3&url=http%3A//w
0x0050   7777 2e73 6563 7572 6974 792d 666f 7275        ww.security-foru
0x0060   6d73 2e63 6f6d 2f66 6f72 756d 2f76 6965        ms.com/forum/vie
0x0070   7774 6f70 6963 2e70 6870 2533 4674 2533        wtopic.php%3Ft%3
0x0080   4431 3431 3332 2532 3673 7461 7274 2533        D14132%26start%3
```

From here I decide to take a quick look at all of the fields in the packet itself, which is proper analysis methodology. This is a key concept that I will elaborate on in the second part of this article series. Inside I see nothing else which could be considered anomalous. Much like the aforementioned syn4k.c there could be another sign in this potential alarm which could help me identify it. Often exploit developers write into their code a telltale sign such as, say, an unchanging TCP sequence number, or other such things to watch for. This is not always the case, but has been known quite often to happen.

Now I build the same filter as above to investigate the other instance of this first alarm, but this time put in the other IP address of 67.18.39.58 (another unremarkable address that was discovered) instead. Then I name the output of this file web_php2 to keep things neat and easy to follow. It only a few seconds of time for the file to be generated. Once again, the "viewtopic.php" signature fragment is found, as seen in the truncated packet below. What I mean by truncated is that as above, for brevity I did not take the entire packets content and paste it here.

```
14:00:05.627398 IP (tos 0x0, ttl 128, id 2631, len 612)
192.168.1.101.1277 > 67.18.39.58.80: P [tcp sum ok]
1602068759:1602069331(572) ack 996286128 win 17520 (DF)
0x0000   4500 0264 0a47 4000 8006 c1f3 c0a8 0165        E..d.G@........e
0x0010   4312 273a 04fd 0050 5f7d a117 3b62 1eb0        C.':...P_}..;b..
0x0020   5018 4470 c924 0000 4745 5420 2f66 6f72        P.Dp.$..GET./for
0x0030   756d 2f76 6965 7774 6f70 6963 2e70 6870        um/viewtopic.php
0x0040   3f74 3d31 3431 3332 2673 7461 7274 3d30        ?t=14132&start=0
0x0050   2048 5454 502f 312e 310d 0a41 6363 6570        .HTTP/1.1..Accep
0x0060   743a 2069 6d61 6765 2f67 6966 2c20 696d        t:.image/gif,.im
```

Now it is apparent that we have two false positive of the "WEB-PHP viewtopic.php access" alarm, so we can move on to investigate the validity of the other alerts in part two of this article series.

5. Concluding part one

In this article we've setup a lab environment that includes the tools we need to analyze an exploit. Then we made use of a publicly available exploit (that uses the old RPC DCOM vulnerability) to test our system, generate IDS alerts, and start looking at the output to determine which are false positive alerts and which are real.

In the second and final part of this article, we'll look at the final four alerts that were generated using our test exploit, and systematically evaluate each one to determine the true nature of our malicious code sample.

About the author

Don Parker Don Parker is an Intrusion Detection Specialist who holds the GCIA certification. He works as an independent consultant and instructor. He also provides other computer security services of a highly specialized nature.