Examining a Public Exploit, Part 2

by Don Parker
last updated September 15, 2004

1. A Recap of Part 1

The first part of this article series set out to create an environment that allowed readers to examine a public exploit as it was sent across the network. The purpose of this exercise is to help the reader understand the complex world of intrusion detection and low-level packet analysis, so that he can better secure his network.

In part one we setup a lab environment using two machines and a set of tools which included Snort, Snortsnarf, Tcpdump, and libpcap (or windump and winpcap for Windows environments). Once the lab was setup, we built an exploit using publicly available source code, and then sent this binary from one test machine to the other. This action triggered an alert with our IDS, Snort, and prompted us to do some low level packet analysis to see what was going on.

With the necessary tools and machines in place, and with our exploit having triggered numerous alerts as shown below, there was only time to discuss the first alert, which turned out to be a false positive (a false alarm). Now we will continue our analysis and analyze the final four alerts that were generated in Snort.

Readers are encouraged to review part one of this article series, to understand how this was generated with snortsnarf, before continuing on.

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

Table 1: IDS alerts generated in part one of this article

2. Continuing the analysis

2.1 "NETBIOS DCERPC ISystemActivator bind attempt" alarm

First I click on the [sid] summary for this alarm as generated in the snortsnarf output, and I see that there is only one source and one destination address for this. Though I am familiar with this alarm I go to the Snort homepage to read quickly once again through the alert's description. Then I can build my bpf filter to isolate all traffic between these two addresses. Though I know that this attack will use TCP as its transport protocol, I still put in "IP" in my bpf filter. If you recall, I do this because I do not want to miss any possible ICMP traffic that may have passed between these two addresses, not to mention any possible UDP for that matter. You are far better off to cast a wide open net, ie., a wide open bpf filter in this case, and then start to tighten it up as you go along. Doing it that way you are more likely to not miss any potential key information such as ICMP echo request packets, or other such lead up activity prior to the attack.

```
C:\ windump.exe -r log_file -nXvSs 0 ip and host 192.168.1.100 and host
192.168.1.101 > dcerpc
```

To quickly describe what the above does, I am telling "windump.exe" to read from the binary log file called "log_file" and look for any packet with a valid IP header in it which also contains the IP addresses as noted above. Additionally, the command is also to dump the findings of the bpf filter to a text file *decrpc*. If we were pressed for time we could put in a bitmask telling windump to give us only PSH/ACK packets. This would result in only getting packets which have a payload. That way, we could quickly see if there truly was exploit code pushed across. Should you wish to write such a bpf filter and bitmask it would look like the one below:

```
C:\ windump.exe -4 log_file -nXvSs 0 ip and host 1921.68.1.100 and host 192.168.1.101 and
tcp[13] = 24 > dcerpc_pshack
```

In this case however I don't tack on a bitmask and the resulting file is bigger then the earlier ones so I open it up using Microsoft Word and simply do a find for the string noted on the Snort website "A001 0000 0000 0000 C000 0000". I find a hit in the first few packets of my file "dcerpc".

```
14:02:07.555843 IP (tos 0x0, ttl 64, id 3301, len 124)
192.168.1.100.1075 > 192.168.1.101.135: P [tcp sum ok]
2869004824:2869004896(72) ack 1636075803 win 5840 <nop,nop,timestamp
5000143 0> (DF)
0x0000   4500 007c 0ce5 4000 4006 a97d c0a8 0164        E..|..@.@..}...d
0x0010   c0a8 0165 0433 0087 ab01 8a18 6184 891b        ...e.3......a...
0x0020   8018 16d0 220b 0000 0101 080a 004c 4bcf        ...."........LK.
0x0030   0000 0000 0500 0b03 1000 0000 4800 0000        ............H...
0x0040   7f00 0000 d016 d016 0000 0000 0100 0000        ................
0x0050   0100 0100 a001 0000 0000 0000 c000 0000        ................
0x0060   0000 0046 0000 0000 045d 888a eb1c c911        ...F.....]......
0x0070   9fe8 0800 2b10 4860 0200 0000                  ....+.H`....
```

This is only a partial match on the signature shown at the Snort homepage, however. Realizing that this is a high threat alarm, if it is indeed valid, I decide to do a bit more research. Looking further at the ASCII content of the packet I use the "F.X.N.B.F.X" and the "MEOW MARB port 135" strings to Google with. Several good hits come back, and one especially from DShield. By verifying the information in that link I am able to positively say that what we found is a true instance of this alarm. As we now realize, this is not good -- someone has now gained full access to the computer with IP address 192.168.1.101. This could also account for the other alarms we saw on the index page in our snfout.alert.ids directory, as was noted in Table 1, above.

At this point I would like to point out how important it is for intrusion detection analysts to try and "game out", or actually use these exploits in a lab environment so that they can see exactly how they look. I have seen how the exploit used in this article looks and works in my home lab, and I can definitively say that this is indeed the RPC DCOM exploit MS03-026.

To be able to truly defend your networks you must also know how to attack them. How can you recognize certain strings which are indicative of a specific attack if you have not recreated it yourself? Of course, it is impossible to study all of the exploit code out there, but it is important to try and keep learning -- this is what will separate the average intrusion detection analyst from a very good one.

2.2 "TFTP Get" alarm

We will now move on to investigate the "TFTP Get" alarm and try to ascertain its validity. Also we will see if it ties into the "NETBIOS DCERPC" alarm that we have just looked at. Before moving on to the actual analysis of the packets in question, we can already believe it is rather likely that it will be a positive alarm. Why? Using the TFTP protocol is a favorite means of transferring files back and forth on compromised machines by malicious hackers. All you need to do this transfer is a TFTP server which is both simple to get and easy to use. Also, you need a TFTP client on the system you are transferring files to and from. Note that Microsoft win32 has a built in TFTP client. If you have never seen or used TFTP and you are using a Windows based machine, simply bring up a DOS prompt and type in "tftp". TFTP is a very neat protocol which you may want to spend some time studying it.

One of the main attributes of TFTP is its speed. It uses UDP as its transport protocol instead of TCP, which is used by FTP. You may recall that a standard TCP header is 20 bytes while a standard UDP header is only 8 bytes. Having 12 fewer bytes makes a big difference in speed. If you are a malicious hacker, speed is often of the essence if you are ferrying files to or from a compromised computer.

Let's now look at the packets that actually triggered the "TFTP Get" signature. By having looked at the TFTP protocol briefly, we now know that it used UDP as its transport protocol. How does this help us? Well, it helps us build a far more accurate bpf filter. We can tell windump or tcpdump that instead of us wanting all packets containing a valid IP header, we want only those packets that contain a valid UDP header. This should cut down on the size of the file as it will no longer any TCP headers. Let's refine our bpf filter to look as follows:

```
c:\ windump.exe -r log_file -nXvSs 0 udp and host 192.168.1.100 and host
192.168.1.101 > tftp
```

Contained in the text file "tftp" is the actual TFTP traffic we are seeing, consisting of the requests to the malicious hackers machine made from the compromised machine 192.168.1.101. Seen below is a snippet of the packets contained in the "tftp" file:

```
14:02:33.701513 IP (tos 0x0, ttl 128, id 4694, len 47)
192.168.1.101.1331 > 192.168.1.100.69: [udp sum ok]  19 RRQ
"evil_file1"
0x0000   4500 002f 1256 0000 8011 a44e c0a8 0165        E../.V.....N...e
```

```
0x0010   c0a8 0164 0533 0045 001b afe6 0001 6576        ...d.3.E......ev
0x0020   696c 5f66 696c 6531 006f 6374 6574 00          il_file1.octet.
```

As we can see in the above file, 192.168.1.101 is requesting the file *evil_file1* from the malicious hacker's machine, 192.168.1.100.

```
14:02:33.702853 IP (tos 0x0, ttl 64, id 3417, len 123)
192.168.1.100.1024 > 192.168.1.101.1331: [udp sum ok] udp 95 (DF)
0x0000   4500 007b 0d59 4000 4011 a8ff c0a8 0164        E..{.Y@.@......d
0x0010   c0a8 0165 0400 0533 0067 d8d7 0003 0001        ...e...3.g......
0x0020   5468 6973 2069 7320 7468 6520 6669 7273        This.is.the.firs
0x0030   7420 6269 6e61 7279 2074 6861 7420 7468        t.binary.that.th
0x0040   6520 6861 636b 6572 2074 7261 6e73 6665        e.hacker.transfe
0x0050   7272 6564 206f 7665 7220 746f 2079 6f75        rred.over.to.you
0x0060   7220 6e6f 7720 636f 6d70 726f 6d69 7365        r.now.compromise
0x0070   6420 6d61 6368 696e 650a 0a                     d.machine..
```

Noted above is the actual transfer of the file *evil_file1*. This file could actually contain anything from flat ASCII to a binary package. The survey kits, or root kits often used by malicious hackers are as varied as the hackers are themselves. What is transferred over the network will largely depend on their aim -- perhaps using the now-compromised box as a jumping point to further probe and attack the internal network. But there are also many other possibilities. It is this long list of possibilities of what a malicious hacker can now do to your computer which will force most people to completely format the hard drive and reinstall.

You may never find all of the potential malware that was copied over onto the computer. It can be hidden in the most obscure places and lay dormant. We now have proof that exploit code was sent to our machine. Unless you are willing to gamble on the security of your network, at this point you would do well to simply format the drive and reinstall all of the packages which were on it. After all, this is why we are all diligent about keeping backups.

2.3 "Attack Responses directory listing" alert

We now have only two remaining alarms to check out, "Attack Responses directory listing" and "Attack Responses 403 Forbidden". We will look at the "directory listing" one first. As we can see via our snortsnarf output, this alert was triggered three times by 192.168.101.

The first packet that triggered this alert has a timestamp of 14:02:13.164600. We can now quickly go to that specific packet and take a look at it. Bearing in mind that the binary file itself could be quite large in real life, we should really create a bpf filter. This filter should give us only the contents of the conversation between the two IP addresses as noted in this alarm:

```
C:\ windump.exe -r log_file -nXvSs 0 ip and host 192.168.1.100 and
192.168.1.101 > responses
```

If you wanted to only do a quick and dirty analysis, then I would simply tack on a bitmask at the end of the bpf filter to show you only the PSH/ACK packets. You may remember that in the PSH/ACK packets is where the actual payload or data will be transferred. By cutting out only the PSH/ACK we are not going to see all of the other packets such as SYN and SYN/ACK, amongst other flag combinations. To see just the PSH/ACK packets alone please see the bpf filter as displayed below:

```
C:\ windump.exe -r log_file -nXvSs 0 ip and host 192.168.1.100 and host 192.168.1.101
and tcp[13] = 24 > psh_responses
```

The above filter will drop all packets between 192.168.1.100 and 192.168.1.101 with the PSH/ACK flags set to the text file called *psh_responses*. It will allow us to quickly scan the packets to see what was sent between these two IP addresses. This is a good way to look at things quickly, but as mentioned earlier in the article you should really cast a wider net to begin with. Once you have an overall feel for what is going on in your log file, you can then narrow your search via a bitmask or fine tune your bpf filter.

By looking at the file that contains only PSH/ACK packets, we can see that the attacker did a directory listing, which is followed by the listing of the directory itself. Continuing through the remaining occurrences of this alarm we can see the same directory listing done for each instance.

We have determined that this alarm is indeed a true occurrence, and not a false positive. Let's move on to the final alarm.

2.4 "Attack Responses 403 Forbidden" alarm

There is one remaining alarm, the "Attack Response 403 Forbidden". The source address for this alarm is 199.60.115.193. I don't recognize this IP address so I use the embedded hyperlink for ARIN to resolve this for me. It resolves to an electronics retail store.

Now we will build a bpf filter such as the one indicated above, with the same bitmask, so it gives us only the PSH/ACK packets. It only takes several seconds to do this. If you are following along, you will then note that the resulting file is still quite large. It would be a pain to go to the specific packet with the timestamp in question. How can we narrow down our bpf filter and bitmask even further so it gives us only what we want?

In the snortsnarf output for this alarm, we can see that only the "ACK" flag is set. The ACK flag has a decimal value of 16 in the 13th byte offset from zero in the TCP header. with that in mind we build a new bitmask of tcp[13] = 16 and just tack that onto the same bpf filter and get windump to process it. Looking at this new file we note that it is still rather large and unwieldy. Once again, how do we prune the results of this to a manageable size? Let's go back and look at the packet that actually fired off this alarm signature:

```
[**] [1:1201:7] ATTACK-RESPONSES 403 Forbidden [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/25-14:04:41.410107 199.60.115.193:80 -> 192.168.1.101:1337
TCP TTL:107 TOS:0x0 ID:44337 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0xAF842A90 Ack: 0x63D099C8 Win: 0xFC67 TcpLen: 20
```

I would now include more information in our filter to further prune the output file. It would seem logical to perhaps take the destination address of 192.168.1.101 and use the ephemeral port of 1337 to further filter the output. It would not make sense to use the source address of 199.60.115.193, server port 80, because this is what may be handing back all of the requested resources. With this train of thought in mind, let's build our new and improved bpf/bitmask:

```
C:\ windump.exe -r log_file -nXvSs 0 ip and src host 199.60.115.193 and dst host
192.168.1.101 and dst port 1337 and tcp[13] = 16 > new_output
```

Once windump has finished generating this file output we can see that the resulting file is much smaller. It is therefore far more manageable for analysis purposes. Now we can go and check the actual packet that triggered this final alert, which has a timestamp of 14:04:41.410107.

I open my output file from the refined bpf/bitmask filter and note that the very first packet in that file is indeed the packet which triggered the alert. Very nice! As you can see it really pays off to know how to write complex filters. You might otherwise have been swimming through a rather large sea of packets just to get to the one of interest.

What happened is that the IP address of 192.168.1.101 requested a resource from the web server at 199.60.115.193, for which it is not allowed access. That is what HTTP error code 403 means, unauthorized. Now that I have tracked this alarm down and found out what it is about, I am not overly concerned. Were it the other way around I would be more interested in what exactly was going on.

3. Wrapping it up

This concludes the analysis part of the article. Hopefully you now realize the value of writing proper filters to help you in your analysis. Whether those filters are Ethereal, tcdump/windump, or other protocol analyzer filters, I would still counsel you to use a tool such as windump, or tcpdump as it forces you to keep your skills up. Protocol analyzers will give you all the answers you need, with little to no effort on your part. However that is not always a good thing, as knowing TCP/IP at a granular level is absolutely essential to truly examine an exploit.

3.1 Building a home lab, and a methodology

To help aid in your analysis, here are my personal recommendations for what you would want in a small home lab, and what your analysis methodology should be. I used to think that a computer lab was a large mass of computers. While it can be that, normally for people learning about intrusion detection, exploits and the link, by wishing to further their knowledge you do not need much more than two or three computers and a switch and/or router.

I would recommend three computers, and they need not be high end ones for that matter either. Several Pentium IIIs or their equivalent are quite adequate to run Win2K XP/2003. On these computers I would set up a dual or triple boot environment to maximize the computer assets. After all, you may as well run as many operating systems as you can on one computer. Not everyone likes to use VMWare and it also has its shares of quirks; a machine needs a lot of RAM to run VMWare and it does not always run well on all computers. Lastly most Linksys, DLink or other personal routers also do double duty as a switch. This allows you to target one of your lab machines from another lab machine with exploit code, or you can use an automated tool. To sum up, I would have one main machine which would be for personal use, along with two dedicated lab machines, and all interconnected via a router/switch.

Finally, it is very important that you approach every packet trace (via a log file) with the same set of steps. Why? By using a common methodology you are far less likely to miss

some critical clue in the packets that you are analyzing. Many people, when analyzing a log file, go immediately to the ASCII content of it. In other words, they look at what is on the right hand side of the viewable packet. This can be a potentially fatal mistake when analyzing a possible attack. Not every exploit will have "/bin/sh" or other telltale sign in the ASCII content. You must get into the habit of approaching your analysis in a logical order. For instance, start at the timestamp and work your way from right to left. Then work your way through the header metrics, and after which look at the hex content. Once this is done, look at the ASCII content for any further possible correlation. And then finally, after all that is done I would then take an overall look at the offending IP addresses in question. Are there any nuggets of info such as repeating TCP sequence numbers and so on? By using a standardized approach to packet analysis you are far less likely to miss something, which could be key to your investigation.

4. Conclusion

I sincerely hope that this article series has given you an insight into packet analysis. Should you wish to contact me over the contents of this article, please feel free to do so. Remember that this was just a high level overview and not a detailed analysis. It will help to serve as a good starting point should this field of endeavor be of interest to you.

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.