

GCIA Practical Assignment

Additional Chapters

The Big Barnyard

Raffael Marty, CISSP
<ram@[cryptojail.net|arcsight.com]>
Mountain View, CA 94041

27th November 2004

Chapter 3

Additional Chapters

3.5 P2P Activity

We discovered that peer to peer activity was the most frequent type of traffic that generated snort alerts. We decided to analyze this traffic, figure out whether these alerts are true positives and whether any other snort alerts were generated by the underlying traffic, indicating false positives. We were also interested in the most involved machines for this type of traffic. If there is a machine in the internal network serving content to a peer to peer network, we suggest taking action and making sure the content offered is not illegal (music, videos, pictures). We will see that quite a few machines are involved in peer to peer traffic and we recommend taking action to eliminate or at least mitigate the problem.

3.5.1 Traffic Summary

In the list of distinct snort alerts found, the following ones are related to peer to peer traffic:

Count	Alert
21102	P2P GNUTella client request
21082	P2P Outbound GNUTella client request
20714	P2P Inbound GNUTella client request
1470	P2P Napster Client Data
10	INFO Outbound GNUTella client request

3.5.2 Involved Machines

There are 11431 distinct destination IPs involved in peer to peer traffic¹ As a next step we were curious whether these machines were targets of any other type of traffic. Therefore, we queried for all the snort alerts targeting the same machines as the peer to peer traffic:²

```
Count  Alert
21092  P2P GNUTella client request
6240   NONE
1470   P2P Napster Client Data
415    SCAN nmap TCP
302    SHELLCODE x86 NOOP
76     SHELLCODE x86 NOOP
75     SHELLCODE x86 inc ebx NOOP
33     BAD-TRAFFIC data in TCP SYN packet
29     DNS zone transfer TCP
25     SCAN SOCKS Proxy attempt
22     SHELLCODE x86 setuid 0
21     WEB-CLIENT readme.eml autoload attempt
20     X11 outbound client connection detected
17     SCAN Squid Proxy attempt
17     SCAN Proxy Port 8080 attempt
12     SHELLCODE x86 setgid 0
12     SCAN FIN
11     CHAT IRC nick change
11     (http_inspect) WEBROOT DIRECTORY TRAVERSAL
6      RPC rstatd query
5      X11 xopen
3      BAD-TRAFFIC tcp port 0 traffic
2      Virus - Possible MyRomeo Worm
2      SCAN synscan portscan
2      CHAT IRC message
1      ATTACK-RESPONSES id check returned userid
1      CHAT AIM receive message
1      SHELLCODE x86 stealth NOOP
1      Virus - SnowWhite Trojan Incoming
1      SHELLCODE x86 0x90 unicode NOOP
1      WEB-MISC http directory traversal
```

¹Remember that we determined how to discover the real sources and targets of an event in the last Chapter. What we found was that P2P connections were utilizing strange port numbers that make it hard to determine what the real source and target was. However, we assume that the higher port number is the source port. This seems to work fine.

²`select distinct(snort_alert) from sans where destip in (select distinct(destip) from sans where snort_alert like "%P2P%" or snort_alert like "%GNUTella%" and service=2 order by destip)`

Quite an elaborate list of alerts and interestingly enough, they represent quite severe attacks. We will have to further analyze these alerts to see whether some of them are false positives. To understand whether any of the above alerts are actually a side effect of the peer to peer activity, we ran three queries. The first one shows *all* the other alerts between machines that triggered P2P rules:

```
CHAT IRC nick change
SHELLCODE x86 NOOP
X11 outbound client connection detected
```

The "X11 outbound client connection detected" is most likely a false positive. The snort rule

```
alert tcp $EXTERNAL_NET 6000:6005 -> $HOME_NET any (msg:"X11 outbound client
connection detected"; flow:established; reference:arachnids,126; classtype:
misc-activity; sid:1227; rev:5;)
```

merely triggers on established flows with target ports between 6000 and 6005³. The "SHELLCODE x86 NOOP" alerts showing up (26 total) are also false positives. Comparing the ports and IPs shows that the rule triggered on a P2P response packet, which does unlikely contain a buffer overflow attempt.

Alert	Source	Destination
P2P GNUTella client request	24.165.202.171:9533	-> 207.166.87.157:61498
SHELLCODE x86 NOOP	207.166.87.157:61498	-> 24.165.202.171:9533

The CHAT IRC nick change event is most likely associated with the peer to peer traffic as well and does not have anything to do with IRC. The loose definition of the rule suggests so:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 6666:7000 (msg:"CHAT IRC nick change";
flow:to_server,established; content:"NICK "; offset:0; classtype:policy-violation;
sid:542; rev:10;)
```

The rule only looks for target ports in the range of 6666 to 7000 and the string "NICK " in the traffic. This can be triggered very easily by for example email traffic⁴.

Another interesting query was to see other events targeting the same target ports and target machines as the peer to peer events:

```
SCAN FIN
SCAN nmap TCP
```

³This is a pretty bad rule if you do not use the `EXTERNAL_NET` and `HOME_NET` parameters. It will trigger on a lot of packets with source ports in the defined ranges.

⁴It would help to have the `HOME_NET` variable set correctly to restrict the number of flows the rules are looking at. As a result the number of false positives would decrease

As other people have found[1], the SCAN FIN is most likely a false positive. The target ports used are too static (i.e., a FIN scan would try to check for multiple ports on a machine and not target only one specific port). Ian Martin[1] suggests that the SCAN FIN alerts are related to the end of a GNUTella sessions. He also mentions that the SCAN nmap TCP alerts are related to GNUTella PUSH events. The reasoning about the SCAN FIN alerts is not entirely sound. All these events originate at sources that are not part of any other traffic. Why would these events never come from an IP that also generated a P2P event? I do not agree that this traffic was ever part of a legitimate GNUTella connection. However, it is hard to exactly determine what this traffic relates to without having the full packet captures. Due to the lack of further proof, I have to let this issue go.

Depending on corporate policies, peer to peer activity is a policy breach⁵. This is a list of all the most active internal machines involved in peer to peer activity:

IP	Count	IP	Count
148.63.156.197	7	115.74.249.65	818
148.63.246.0	16	32.245.166.236	4005
217.159.17.213	20	207.166.87.157	8342
170.129.50.120	377	138.97.18.88	8636

From a legal standpoint, the distribution of music, videos, software and alike, is illegal. Offering this content via peer to peer services could result in legal action against the company. The following machines are offering content and should be encouraged to turn their services off:

IP	Count	IP	Count
115.74.249.65	1	32.245.204.168	9
207.166.87.157	1	115.74.33.251	10
32.245.166.236	1	32.245.214.22	31
170.129.50.120	7	207.166.135.150	281

We recommend blocking peer to peer traffic on the border routers. We are aware of the difficulties related to this. Peer to peer traffic could be tunneled through HTTP traffic and is therefore a nightmare to filter out. It might make sense to use a proxy server for the traffic, which greatly facilitates the task of eliminating peer to peer traffic. If this is an academic environment, where blocking traffic or deploying proxies is not a solution, stringent policies and their enforcement could be used to address the problem. We would further recommend to educate users about the associated risks with downloading illegal content via P2P. The risk of virus infections, trojan horses, spyware and alike is getting higher and higher, let alone the legal consequences, if a user is caught for offering illegal content via one of the peer to peer networks.

⁵A good corporate policy should definitely prohibit this activity and it should be stringently enforced!

3.6 TTL Investigations

During our investigations, we came across some very strange packet captures:

```
09:10:18.224488 IP (tos 0x0, ttl 125, id 11867, offset 0, flags [DF],
length: 331, bad cksum ff8b (->a0cb!)) 138.97.18.88.64920 > 64.154.80.51.80:
P [bad tcp cksum 49 (->4089!)] 165783487:165783778(291) ack 603624956 win 8760
```

```
09:10:18.284488 IP (tos 0x10, ttl 240, id 0, offset 0, flags [none],
length: 1500, bad cksum ff00 (->9785!)) 138.97.18.88.64920 > 64.154.80.51.80:
P [bad tcp cksum 0 (->166a!)] 437841469:437842929(1460) ack 3857125828 win 33580
```

Both entries have the same timestamp; they are from the same source machine and port and go to the same machine and port. However, the IP IDs and the TTLs are completely different. Looking further into the IP addresses involved in this traffic, we found that 138.97.18.88 is not assigned to anybody⁶ and 64.154.80.51 resolves to hitbox.com, a Web analytics company. The source address is a machine we already came across during the analysis of automated behavior (see Section 3.2). This, along with the GCIA practical from Richard A. Baker[2] lets us believe that this is some kind of a scripted attack involving spoofed source addresses. Possibly this is a DoS attack as suggested by Richard. However, the event volume is very low and therefore unlikely to be a DoS. There are a total of 7739 events generated during a period of seven days. It is possible that many more packets were present on the network but did not trigger any snort alerts. It is also interesting to note that the traffic is not HTTP, although the destination port is 80. Either this is some kind of tunneled traffic, absolutely random payload or as we discussed in Section 3.2, this could be images or compressed content.

To find whether there was any more of this type of traffic, we did the following analysis:

1. Run a query on the database to return all the tuples having the same connection information and TTL: `select sourceip, destip, sourceport, destport from sans group by sourceip, sourceport, destip, destport, ttl order by sourceip, destip, sourceport, destport, ttl`
2. Trim the last column, which is the TTL.
3. Get all the unique entries in the data set, sort and eliminate everything that only showed up once: `awk '{print $1,$2,$3,$4}' | uniq -c | sort -rn`

The idea behind this is that we first get all the distinct five-tuples of IPs, ports and the TTL. Eliminating everything with a count of one and eliminating the TTL field, shows all the entries that had two different TTLs. This analysis resulted in the following list of source IPs:

⁶`whois -h whois.ripe.net 138.97.18.88`

Count	IP	Count	IP
2564	115.74.249.65	2	209.249.111.2
3	128.167.120.13	1	210.220.73.27
3272	138.97.18.88	1	211.21.238.234
1	167.216.198.217	3	216.100.239.56
1	167.79.91.3	4	216.79.39.204
1942	170.129.50.120	12466	32.245.166.236
7	170.140.88.160	1	61.113.110.123
4	170.140.90.134	2	63.211.17.228
7	193.170.68.68	2	63.236.43.103
2	193.41.181.254	1	66.157.162.54
4	195.25.184.195	1	66.48.23.187
1	204.188.136.123	1	68.15.105.8
1	204.253.57.44	3	80.208.98.134
8418	207.166.87.157	1	80.56.69.75
1	207.89.144.113	22	81.80.75.34
3	208.45.79.122		

This list is much too long. Traffic from the same sources with different TTLs should never show up, unless someone is messing with the TCP/IP settings on their machines. The target ports used by these events show mainly HTTP traffic:

Count	Port	Count	Port
5	53	1	63238
28732	80	1	64850
1	62045	1	64943
1	62046		

There are 406 targeted machines. These are the ones targeted the most:

Count	IP	Count	IP
5001	64.154.80.51	1641	64.154.80.45
2278	64.154.80.50	1426	64.154.80.47
1770	209.11.34.129	1407	208.33.48.102
1738	64.154.80.49	1192	208.33.48.101
1642	64.154.80.44	787	209.11.34.130

We do not know why this traffic shows up in the log files. It could be that the obfuscation has to do with this phenomenon. We would need more information in order to solve this puzzle.

Bibliography

- [1] *Ian Martin - GCIA practical*
http://www.giac.org/practical/GCIA/Ian_Martin_GCIA.pdf.
- [2] *Richard Baker GCIA Practical*
http://www.giac.org/practical/GCIA/Richard_Baker_GCIA.rtf.