

Terminal--Snort Challenge (Part 3)

Solution (Regular expressions)

The basic regex that we need for one character is

```
[0-9,A-F]
```

This specifies a range of possible values for the character. It can be any digit 0 through 9 or any letter A through F. Normally a character set for hex characters also includes lower case letters a through f, but those are not present in our packet captures.

We specify the number of consecutive characters we want to match using curly braces { }. If we want to match a string when it reaches 24 characters, we would use {24}. A string between 20 and 30 characters would be {20,30}. If we use {24} and the string is longer, that is fine; the rule will fire when it sees 24 characters and ignore the rest.

A regex to match a string of at least 24 hex characters (upper case letters) would be

```
[0-9A-F]{24}
```

The choice of 24 characters was arbitrary. The packets we saw had hex strings 0x/26 (decimal 38) characters long, but we may not have seen all the possible packets.

Snort rules

The basic Snort rule syntax is [explained in this pdf](#). Normally the header looks something like this:

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any.
```

The strings \$EXTERNAL_NET, \$HTTP_PORTS and \$HOME_NET are variables that are configured in /etc/snort/snort.conf when Snort is installed; they specify external and internal IP addresses and ports related to services like web. This rule would be looking for traffic from outside web servers coming in to our network. We will not need to be that specific, although it is good practice. We will write a quick and dirty, ugly rule to solve the terminal. There is an appendix to this document with a more reasonable rule.

The most generic header we could have would be this.

```
alert udp any any <> any any
```

That selects traffic with any IP address and port going in any direction. The only thing it looks for is the UDP protocol. We can do a little better than that; we know one of the ports will be 53.

```
alert udp any 53 <> any any
```

The body of the rule follows the header. It is enclosed in parentheses, and the parts are separated by semicolons. Most rules have a message, so we can use this.

```
msg: "DNS--wannacookie cnc detected";
```

We will skip the Flow option, since that applies to TCP traffic.

Snort uses Perl Compatible Regular Expressions, or PCRE. The detection option for a regular expression is pcre: . Additionally, the pcre is enclosed in quotes and / characters. This will be the heart of our

rule.

```
pcrc: "[0-9,A-F]{24}"/;
```

Finally, all rules must have a Signature ID. The custom is to use ID numbers of 1,000,000 or higher for local rules, that are not part of the official Snort rule set.

```
sid: 1000001;rev:1;
```

To put all together, our rule is

```
alert udp any 53 <> any any (msg: "DNS--wannacookie cnc detected";  
pcrc: "[0-9,A-F]{24}"/; sid: 1000001;rev:1;)
```

Please note that this is a very sloppy rule. The pcre test consumes a lot of processor time so normally there is a content check first, to make sure that we are dealing with a DNS packet before the pcre executes. Another Kringlecon player told me, "If I wrote a rule like that at work I'd be fired."

Put the rule to work

Enter the Snort terminal and follow the instructions on the main page and the moreinfo.txt file. Use a text editor to place the rule in the file, `/etc/snort/rules/local.rules`. Then start Snort with this command.

```
snort -A fast -r ~/snort.log.pcap -l ~/snort_logs -c  
/etc/snort/snort.conf
```

This tells snort to alert on traffic and use fast (brief) logging (`-A fast`). Rather than copying traffic from a network interface, it will read the file `snort.log.pcap` in the user's home directory. It will place the log file and a pcap containing captured packets in `~/snort_logs`. Finally, it will read the Snort configuration file in `/etc/snort/snort.conf`.

Troubleshooting

Snort is very finicky about rule syntax, so do not be surprised if Snort does not start on the first attempt. The terminal tells you whether you completed the challenge even if you don't run Snort(!). When Snort runs, it posts pages of information to the terminal. If you have syntax errors, the error messages will appear at the very end of the output. Beware of Windows "smart quotes", as they cause problems.

Once you have syntax errors corrected, look at `~/snort_logs/alert` to see what the rule alerted on. You must catch traffic in both directions to get credit for the terminal. You can also use `tcpdump` or `tshark` on the terminal to look at the packet capture (in `~/snort_logs/`) if you need more information about the packets that your rule alerted on.

If you want to run a local copy of Snort, be aware that the installation is complicated. There is better support for Snort on CentOS or Fedora, so use that distribution.

Better Rules

It would be better to split the rule into two, one for inbound and one for outbound traffic. Then we can use the `$HOME_NET` and `$EXTERNAL_NET` variables to limit the packets we examine. The new rules also include a content check for the DNS flags that specify query or response.

In this query, note that there are two bytes of flags equal to 01 00 hex. Also, the bytes are found two bytes after the start of the DNS payload. The first two bytes hold the Transaction ID (0x94e3 for this packet), and the next two hold the flags.

```

✓ Domain Name System (query)
  Transaction ID: 0x94e3
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  > Queries
    [Response In: 2]

```

0000	45 00 00 5f 00 01 00 00 40 11 99 e3 0a 7e 00 c8
0010	f9 0d dc 56 aa 56 00 35 00 4b 64 c6 9e 43 01 00
0020	00 01 00 00 00 00 00 00 26 37 37 36 31 36 45 36
----

This phrase will look for 01 00 hex in the second and third bytes of the payload. It skips two bytes (offset: 2) and then takes the next two bytes (depth: 2).

content:"|01 00|"; offset:2; depth:2;

For response packets, the flags look like this.

```

> Frame 2: 159 bytes on wire (1272 bits), 159 bytes captured (1272 bits)
> Internet Protocol Version 4, Src: 249.13.220.86, Dst: 10.126.0.200
> User Datagram Protocol, Src Port: 53, Dst Port: 43606
✓ Domain Name System (response)
  Transaction ID: 0x94e3
  > Flags: 0x8400 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0

```

0000	45 00 00 9f 00 01 00 00 40 11 99 a3 f9 0d dc 56	E.....@.....V
0010	0a 7e 00 c8 00 35 aa 56 00 8b 91 43 9e 43 84 005.V...C.C.
0020	00 01 00 01 00 00 00 00 26 37 37 36 31 36 45 36877616E6
0030	45 36 31 36 33 36 46 36 46 36 42 36 39 36 35 32	E61636F6 F6B69652
0040	45 36 44 36 39 36 45 32 45 37 30 37 33 33 31 06	E6D696E2 E707331.

This phrase will tell us we have a DNS response.

content:"|84 00|"; offset:2; depth:2;

These rules are more specific and will do a simple check to ensure the packet is DNS before executing the expensive pcre check.

```

alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg: "DNS--wannacookie
cnc detected outbound"; content:"|01 00|"; offset:2; depth:2;
pcre:"/[0-9A-F]{24}"/; sid: 1000002;rev:2;)

```

```

alert udp $EXTERNAL_NET 53 -> $HOME_NET any (msg: "DNS--wannacookie
cnc detected inbound"; content:"|84 00|"; offset:2; depth:2;
pcre:"/[0-9A-F]{24}"/; sid: 1000001;rev:2;)

```

```
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# -----
# LOCAL RULES
# -----
# This file intentionally does not come with signatures.  Put your local
# additions here.

alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg: "DNS--wannacookie cnc detected outbound"; content:"|01 00|";offset:2;depth:2; pcre:"/[0-9A-F]{24}/"; sid: 1000002;rev:2;)

alert udp $EXTERNAL_NET 53 -> $HOME_NET any (msg: "DNS--wannacookie cnc detected inbound"; content:"|84 00|";offset:2;depth:2; pcre:"/[0-9A-F]{24}/"; sid: 1000001;rev:2;)
~
~
~

elf@4ab60b545b11:~$ vi /etc/snort/rules/local.rules
elf@4ab60b545b11:~$
[+] Congratulation! Snort is alerting on all ransomware and only the ransomware!
[+] 
```

Up Next

We can gain useful intelligence about the ransomware if we analyze the network traffic in the packet capture file. Even better, we get to learn about Wireshark's command line sibling, tshark!