There is a huge performance decrease with /dev/zero traffic and some activated Rules.

Suricata is used in inline mode:

```
suricata -c /etc/suricata/suricata.yaml -q 0
```

The setup is built with 4 machines, 2 clients and 2 servers that connect the 2 clients and on one server is suricata running.

The rules used are:

```
- botcc.rules
- ciamy.rules
- compromised.rules
- drop.rules
- dshield.rules
- emerging-current_events.rules
- emerging-malware.rules
- emerging-mobile_malware.rules
- emerging-scan.rules
- emerging-trojan.rules
- emerging-user_agents.rules
- emerging-worm.rules
```

Some profiling output:

[link to profiling output]

The test is made by creating 2 different testfiles:

```
dd if=/dev/zero of=testfile bs=1M count=4096
```

```
dd if=/dev/urandom of=testfile2 bs=1M count=4096
```

The transfer is made with netcat:

Client A:

```
nc -v -v -l -n -p 2222 >/dev/null
```

Client B:

```
pv -t -r -a -b testfile | nc -v -v -n $IP 2222 >/dev/null
```

The diff between testfile and testfile2 was 160Mbit/s to 40Mbit/s. The same rules within snort don't decrease the performance like that.
I suspect that one or more of these rules have \[00\] or \[00 00\] as a fast pattern. As each data byte of the stream is 00, it will trigger the more expensive 'match'-patch for each packet (and even more often in the AC matcher I think).

Andreas Herz wrote:

I parsed several of the rules (see pastes) that were called and didn't see that pattern.

In the meantime I could also confirm the test via ftp also has the 4:1 difference, but downloading the files via http didn't show such a 4:1 difference.

I don't understand why it's so much better with HTTP:

\[
\text{while true; do } \{ \text{echo -e 'HTTP/1.1 200 OK\r\n'; cat testfile; } \mid \text{nc -l -p 8000; done}
\]

and

\[
\text{wget http://10.0.20.89:8000/}
\]

Don't have the gap although in the sniffer it's always just zeros \(\backslash r\backslash n\) (except the initial connect). It's also unrelated to the port I use, since I tried also ports not in \$HTTP_PORTS. Using \$HTTP_PORTS for simple tcp connection still result in the decrease. So it looks like that it's decreasing with every tcp connection except HTTP.

In Profiling I also see, that the rules are still read and have kinda the same ticks.

Andreas Herz wrote:

I deactivated emerging-current_events.rules and emerging-scan.rules and now I see no rule ticks in the rule_perf.log but the decrease is still there, so I doubt that it's related to specific rules.
As asked, the packet_stats for "fast" HTTP and "slow" TCP with the same file.

#6 - 07/29/2014 07:52 AM - Andreas Herz
I did the same test with real hardware now and this is what i got:

CPU: Intel(R) Core(TM) i3-3220 CPU @ 3.30GHz
RAM: 2GB
NETWORK: intel e1000e and igb cards

I get 7.5MB/s (60Mbit/s) with the /dev/zero traffic and 48MB/s (384Mbit/s) with the /dev/urandom traffic.

#7 - 07/29/2014 09:19 AM - Victor Julien
Can you try narrowing down the ruleset? It could be that there are just a few rules causing this.

#8 - 07/29/2014 09:33 AM - Andreas Herz
Victor Julien wrote:

Can you try narrowing down the ruleset? It could be that there are just a few rules causing this.

I narrowed it down to "emerging-scan.rules" and "emerging-trojan.rules", with those 2 active it's still the huge gap. Although using only one of them doesn't result in the gap, so it's only the case with both active.

In the next step i would start with deleting some rule block within the files, unless you have a better idea :)
Fast-pattern:
I suspect one of the things that is so costly here is that we store each AC match with an offset. For the 2-byte pattern that would mean we store it payloadlen-1 times. This has to be costly. Also, this mechanism currently doesn't take the depth of the pattern into account. It's probably not easy to have the AC matcher take depth into account, but it I think it would be possible to not store these offsets beyond the depth/dsize.

So your suggestion for now is to take a look at the code? Or is there anything else I could do? Changing the rules for example.

For this signature:

Sid: 2014600
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Win32/Nitol.A Checkin"; flow:from_client,established; dsize:1028; content:"01 00 00 00");

Adding "fast_pattern;" after content:"01 00 00 00"); would use that as the pattern, which is more unique than all zeros.

So:

Sid: 2014600
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Win32/Nitol.A Checkin"; flow:from_client,established; dsize:1028; content:"01 00 00 00"); fast_pattern;
Have you tried rule profiling? That should point out which rules are taking the most time.

This signature should not be firing with all zero data:

Sid: 2018167
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Generic CnC"; flow:established.to_server; content:" Mini BackDoor[00]"; offset:9; depth:20; reference:md5,398b6622a2c8e472a4340d3e79e654b; classtype:trojan-activity; sid:2018167; rev:1;)  
Rule matches on packets.  
Rule matches on reassembled stream.  
Rule contains 1 content options, 0 http content options, 0 pcre options, and 0 pcre options with http modifiers.  
Fast Pattern " Mini BackDoor\x00" on "payload and reassembled stream" buffer.

For this signature:

Sid: 2017935
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Backdoor family PCRat/Gh0st CnC traffic (OUTBOUND) 12 SET"; flow:to_server,established; dsise:8; content:"[00 00]"; offset:2; depth:2; content:"[00 00]"; distance:2; within:2; flowbits:set,ET.gh0stFmly; flowbits:noalert; reference:url, www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Backdoor%2Awf%32%2FMcClient.ZR&ThreatID=-2147325231; reference:md5,3b1abb60bafbab204aedd68acdf58ac9; classtype:trojan-activity; sid:2017935; rev:3;)  
Rule matches on packets.  
Rule contains 2 content options, 0 http content options, 0 pcre options, and 0 pcre options with http modifiers.  
Fast Pattern "x00\x00" on "payload" buffer.  
No warnings for this rule.

The content of ":[00 00]" is not a good filter and on input data that is all zero, it will trigger on all but the first byte.

In this case, the dsise:8 would be a better filter to eliminate packets, as it should check the payload size is 8 bytes.

This signature should not be firing with all zero data:

Sid: 2018167
I thought so, but if i have just this rule active there is still a drop from 120mb/s down to 90mb/s, so it's at least checked.

Adding "fast_pattern;" after content:"[01 00 00 00]"; would use that as the pattern, which is more unique than all zeros.

Perfect, that helped to make this rule much faster, with this addition the performance drop down to 20mb/s (with only this rule active) got back to 120mb/s. I guess i will send this to ET, unless you want since you found this.

I tried but that still brings me to those 4 rules. The rules are not good as we see but still shouldn't suricata try to handle such rules faster? I'm not sure what snort does different with them.

This signature should not be firing with all zero data:

Sid: 2018167
I thought so, but if i have just this rule active there is still a drop from 120mb/s down to 90mb/s, so it's at least checked.

Adding "fast_pattern;" after content:"[01 00 00 00]"; would use that as the pattern, which is more unique than all zeros.

Perfect, that helped to make this rule much faster, with this addition the performance drop down to 20mb/s (with only this rule active) got back to 120mb/s. I guess i will send this to ET, unless you want since you found this.

Ken Steele wrote:

Have you tried rule profiling? That should point out which rules are taking the most time.
I tried but that still brings me to those 4 rules. The rules are not good as we see but still shouldn’t suricata try to handle such rules faster? I’m not sure what snort does different with them.

Yes, please report it to ET. You are welcome to mention my name, since I know several of them.

#20 - 08/05/2014 11:55 AM - Will Metcalf
We can add a set of Nulls preceding the windows match which should improve perf. Seems present in all samples.

#21 - 08/06/2014 02:39 AM - Andreas Herz
Will Metcalf wrote:

We can add a set of Nulls preceding the windows match which should improve perf. Seems present in all samples.

Can you describe this a little bit more :)?

#22 - 01/01/2016 06:22 PM - Andreas Herz
- Assignee set to Andreas Herz
- Target version set to TBD

#23 - 02/20/2016 06:09 PM - Andreas Herz
- Status changed from New to Closed

Diff is still there with 3.0 but much much closer, so IMHO just normal behaviour in this case. Closing for now.

#24 - 02/21/2016 03:02 AM - Victor Julien
- Assignee deleted (Andreas Herz)
- Target version deleted (TBD)

Files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>packet_stats.log.nc</td>
<td>3.67 KB</td>
<td>07/22/2014</td>
<td>Andreas Herz</td>
</tr>
<tr>
<td>packet_stats.log.http</td>
<td>4.48 KB</td>
<td>07/22/2014</td>
<td>Andreas Herz</td>
</tr>
<tr>
<td>zero.rules</td>
<td>69.6 KB</td>
<td>07/30/2014</td>
<td>Andreas Herz</td>
</tr>
<tr>
<td>foobar.rules</td>
<td>1.71 KB</td>
<td>07/30/2014</td>
<td>Andreas Herz</td>
</tr>
</tbody>
</table>