Ok, I am not sure where this goes, but I can not seem to be able to craft a home_net/external_net combination that gives me only ipv6 address space. Is this configurable at all? Is this an issue where home_net/external_net can't be configured to process only ipv6 traffic?

I have encountered what is likely a related bug while porting Suricata 2.0.3 to the pfSense firewall. Here is how to reproduce the problem.

I have a test VM where HOME_NET has both IPv4 and IPv6 addresses. I also tried with only IPv6 addresses, but get the same results. Set EXTERNAL_NET to "!$HOME_NET". I used this very simply rule to test. The failure to alert is repeatable for me.

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"IPv6PING-request"; itype:128; classtype:icmp-event; sid:6000001; rev:1;)
```

Ping6 the Suricata host and you will get no alerts. Change $EXTERNAL_NET to "any" instead, then ping6 and you will get alerts. I enabled debug output and see that while all IPv4 networks in HOME_NET are "negated" when producing EXTERNAL_NET, none of the IPv6 addresses are negated (and they should be).

Some more testing yielded the following additional information about this issue. All testing was done with this simple rule:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"IPv6PING-request"; itype:128; classtype:icmp-event; sid:6000001; rev:1;)
```

Any IPv6 addresses within the $EXTERNAL_NET variable are ignored as "sources". You get no alerts. However, if you change the rule above and set any specific IPv6 address as the source, and then ping6 from that source, you do get alerts. The code farther down is correctly decoding IPv6 addresses when it receives them. The problem seems to be in parsing and decoding IPv6 addresses defined indirectly within the variable. In particular it appears from my limited testing to only occur when the variable is also negated (as when you define $EXTERNAL_NET = "$HOME_NET").

With this definition , no IPv6 address ranges get negated and added to the DetectAddress lists. IPv4 works fine in this situation, though.

I tried every combination I could think of defining EXTERNAL_NET with EXTERNAL_NET = "$HOME_NET", or EXTERNAL_NET = "(!$HOME_NET)" or even EXTERNAL_NET = "(!$HOME_NET)". None of the variations seemed to matter.

Is there any relevant info with your set up in the suricata.log when run in verbose mode (-v)?

Are your network variables defined with the syntax below:

HOME_NET: "[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12,someIPv6:addr:ress]"

EXTERNAL_NET: "$HOME_NET"

I have not tried a run in verbose mode, but will shortly. I did compile a version with debug enabled and have run that. Those logs get rather large, but did help me realize what is happening. I can offer this evidence from my testing thus far:

1 -- I made sure all my addresses (both IPv4 and IPv6) in HOME_NET are specified in CIDR form. For the handful of single endpoint addresses I used /32 or /128 depending on the address family.

2 -- I tried defining EXTERNAL_NET two different ways. If I use the popular "$HOME_NET" form, then I never see any IPv6 address ranges being
I am talking about the negated ranges (or in other words all the IPv6 addresses that are NOT the ones defined in HOME_NET). However, if I specify a specific IPv6 address in EXTERNAL_NET such as EXTERNAL_NET: "!someIPv6Address", then I do see the NOT ranges generated and printed in the debug log.

3 -- Digging around in the "detect-engine-address.c" source code module late last evening I found what I believe is at least the first problem. Around line #987 a "for loop" is used to insert the negated IPv4 addresses from a resolved variable into the "ghn" Address Group list. However, there is no corresponding "for loop" to do the same for any IPv6 addresses resolved from the variable. Initially I thought that was the only bug, but after adding the missing "for loop" like so --

```c
/* insert the IPv4 addresses into the negated list */
for (tmp_ad = tmp_gh.ipv4_head; tmp_ad; tmp_ad = tmp_ad->next) {
    DetectAddressPrint(tmp_ad);
    DetectAddressInsert(NULL, ghn, tmp_ad);
}

/* insert the IPv6 addresses into the negated list */
for (tmp_ad = tmp_gh.ipv6_head; tmp_ad; tmp_ad = tmp_ad->next) {
    DetectAddressPrint(tmp_ad);
    DetectAddressInsert(NULL, ghn, tmp_ad);
}
DetectAddressHeadCleanup(&tmp_ghn);
```

it will now print the resolved IPv6 address ranges in the debug output, but still will not alert on IPv6 when using variables in a rule. The code I added above is the loop for IPv6, the IPv4 loop was existing. I'm thinking there is still another part of the code where perhaps IPv6 addresses pulled from variables are not getting into the necessary address match lists.

This all seems isolated to using variables in rules. If you put a specific IPv6 address in a rule, then the alert works. So instead of using $EXTERNAL_NET as the source in the rule, if you just edit the rule to use any specific IPv6 address or address range, you get the alert.

So to summarize, it looks like IPv6 addresses are not fully resolved and stored into the various match lists when those addresses are part of variables.

Bill

#6 - 09/03/2014 08:22 AM - Erik C

Bill;
I am not sure I follow, but it sounds like you are at the crux of my issue.

If I specify

HOME_NET: "2001:4d0:2310:150::0/120"

and

EXTERNAL_NET: "![[1.2.0.0/16, 1.3.0.0/16]]" (known ipv4 ipspace in the stream)

I can't get just the ipv6 traffic in home_net. What I need is to specify my ipv6 cidr in HOME_NET and get only traffic to and from that ip space, or ipv6 in general. If I can just get all ipv6 and no ipv4 traffic, that would be fine also.

My problem is simple. I have comimgled ipv4/ipv6 traffic on my tap, and I need to use suricata http logging to handle just ipv6. Since Suricata can not handle 10 gig traffic at all (see bug #705), my only solution is to cut out everything that is not ipv6 and log just that, which Suri should, in theory, be able to handle.

If bug #705 would be resolved (as of the most recent beta release, it still isn't), then trying to do this would be unneeded. However, I think discovering a way to properly sniff only ipv4 or ipv6 traffic by manipulating home_net and external_net is just as important for chopping up load between Suri installs depending on throughput of those network types.

Erik

#7 - 09/03/2014 08:24 AM - Erik C

edit: Specifying the EXTERNAL_NET and HOME_NET does not allow me in any way to segregate ipv4 from ipv6 traffic for analysis. This is what I may not have been clear about in the above post.

#8 - 09/03/2014 09:12 AM - Bill Meeks

Erik:

Our issues are different but I think still somewhat related. What I've noticed is that for rules where either one of the source or destination RULE VARS is negated, then IPv6 traffic won't be picked up. For example, specifying EXTERNAL_NET as "$HOME_NET" results in no IPv6 traffic being logged when the source or destination RULE VAR is $EXTERNAL_NET.

We each have a different goal, but I believe the same code bug is behind the malfunctions we each see. If you think our problems are not related, I can bust my report out into a separate issue so as not to interfere with yours. I tagged on to the issue you reported because I initially thought they
My problem is simple. I have comingled ipv4/ipv6 traffic on my tap, and I need to use suricata http logging to handle just ipv6. Since Suricata can not handle 10 gig traffic at all (see bug #705), my only solution is to cut out everything that is not ipv6 and log just that, which Suni should, in theory, be able to handle.

Wouldn't a BPF help here? "ip6" should make sure Suricata only processes IPv6 packets.

I have merged the fixes by Bill Meeks based on https://github.com/inliniac/suricata/pull/1120