SEPTun Mark II

Chapter 1

Introduction

SEPTun is (hopefully at least) a yearly series of articles that guide through the extreme and experimental techniques to deploy Suricata IDS/IPS/NSM in high-speed networks.

In this article, we reflect and share our experience and knowledge about what we have learned - with detailed instructions on how to deploy those findings. Alongside that, we also describe the challenges we hit through that journey.

Each Mark summarizes last year's one and does not include content that's still valid from the last one. If something needs to be updated, we show it here, if not - the old version of documents still hold. Please read SEPTun Mark I if you haven't already before you read this one.

Ready? Fasten your seatbelts and enjoy the ride!!

As we were concluding our research - the big Meltdown happened. One of the major worries with the subsequent kernel patches is that it can affect the performance of 15%+. We did some testing to confirm if that would have an effect on our setup and have documented those in this article as well.

SEPTun Mark II findings were initially presented on SuriCon 2017.

How is this guide different from Mark I?

- eXpress Data Path - describe what it is and how it can be used
- We received numerous questions about RSS. Deployment guide for RSS is introduced in Mark II

Our set up(s)

Kernel

This guide has been tested and confirmed to work well on (we needed something with a current toolchain and kernels):

- Debian Testing (Buster with kernel 4.14.x/4.15.2)

- Ubuntu LTS Xenial (kernel 4.13.10/ 4.15.2)

If you would want to run a test build a Vagrant box is located at the Vagrant Cloud

General set up

- The traffic we tested on was a mix of ISP and corporate type in different setups in the range up to 20Gbit/sec speeds with long and sustained peaks.
- Suricata 4.1.0-dev branch/latest git master at the time of this writing.
- Using Intel NIC X710/X520/X510 (i40/ixgbe) - one port on each card used. Cards installed on separate NUMA nodes.
- Full ETpro ruleset

HW set up 1

- 128GB RAM, 8 DIMMS, 4 per socket.
- 2x Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz - 28 cores total, HT enabled and used for 56 hardware threads.

HW set up 2

- 64GB RAM, 4 DIMMS, 2 per socket.
- 1x Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz - 8 cores total, HT enabled and used for 16 hardware threads.

Haswell is recommended, Sandy Bridge is a minimum.

Chapter 2

What is XDP

XDP provides another Linux native way of optimizing Suricata's performance on sniffing high-speed networks.

" *XDP or eXpress Data Path provides a high performance, programmable network data path in the Linux kernel as part of the IO Visor Project. XDP offers bare metal packet processing at the lowest point in the software stack which makes it ideal for speed without compromising programmability. Furthermore, new functions can be implemented dynamically with the integrated fast path without kernel modification.* "

More info about XDP

XDP advantages

- No specific HW requirements
- Bare metal packet processing
- Integrated fast path in the kernel stack
- Programmable

XDP is not another generic kernel bypass technology. It augments AF_Packet and does not replace it.

XDP bypass for Suricata

Coding began by Eric Leblond (@regiteric) 2 years ago (on the eBPF initially then XDP in 2017) for implementation in Suricata. Peter Manev(@pevma) started preliminary testing in April/May 2017, Michal Purzynski (@MichalPurzynski) joined the test and research effort in September 2017.

XDP allows for bypass of flows that you/Suricata is not interested in after certain size. It does that at the earliest possible stage in the Linux stack. It drops packets before the SKB is built, saving Linux kernel from doing most of the processing only to throw away results later.

To do that XDP depends on eBPF scripts. Those scripts are compiled with JIT and are really fast. Note - you have heard a lot of bad press about eBPF in context of the recent Spectre attacks. We address those concerns in this paper, read on.

You may wonder whats eBPF is and what it has to do with XDP bypass for Suricata.

eBPF

- Improvement over classical BPF
- Allows for user hooks/programs to be run per packet
- Extends and improves performance of Suricata
- Opens more kernel space possibilities
- Elephant flow bypass
- You need to write your own eBPF "hooks"

One of the better descriptions of eBPF is found in the linux manpages:

> In Linux, it's generally considered that eBPF is the successor of cBPF. The kernel internally transforms cBPF expressions into eBPF expressions and executes the latter. Execution of them can be performed in an interpreter or at setup time, they can be just-in-time compiled (JIT'ed) to run as native machine code. Currently, x86_64, ARM64, s390, ppc64 and sparc64 architectures have eBPF JIT support, whereas PPC, SPARC, ARM and MIPS have cBPF, but did not (yet) switch to eBPF JIT support.
>
> eBPF's instruction set has similar underlying principles as the cBPF instruction set, it however is modelled closer to the underlying architecture to better mimic native instruction sets with the aim to achieve a better run-time performance. It is designed to be JIT'ed with a one to one mapping, which can also open up the possibility for compilers to generate optimized eBPF code through an eBPF backend that performs almost as fast as natively compiled code. Given that LLVM provides such an eBPF backend, eBPF programs can therefore easily be programmed in a subset of the C language. Other than that, eBPF infrastructure also comes with a construct called "maps". eBPF maps are key/value stores that are shared between multiple eBPF programs, but also between eBPF programs and user space applications.

The most significant benefit is that for XDP bypass the Linux kernel does not have to create the SKB (Linux socket buffer structure) - hence saving CPU cycles on creating something that would be thrown away later anyway. Not doing unnecessary work improves performance. Decide early, drop early.

There are three modes available (af-packet section in the suricata.yaml config):

- `xdp-mode: soft`
- `xdp-mode: driver`
- `xdp-mode: hw`

The "hw" mode means packets will never be seen by the Linux kernel and will be dropped at the card itself, at the hardware level. Intel cards cannot do it as of now (the writing of this article). Netronome cards can do it.

The "driver" mode means packets will be dropped before the SKB is created, at the driver level, but not in hardware. Linux kernel will see whose packets but will drop them very early saving most of the processing time. Intel cards with drivers from the Linux kernel support it. The upstream Intel version of those drivers (from SourceForge) do not support XDP and it's unlikely they ever will.

The "soft" mode means packets will be dropped after the SKB is created and before Suricata can consume those packets. The "soft" mode is the slowest one but also an excellent fallback since it does not need any hardware or driver support.

In this article, we use `xdp-mode: driver` for cards that support eBPF.

NICs with native driver XDP support

- Broadcom
- Cavium/Qlogic
- Cavium
- Intel: ixgbe + i40e
- Mellanox
- Netronome
- Virtio-net

`xdp-mode: hw` is available only in Netronome currently at the time of this article writing.

Prerequisites

Please note, that symmetric RSS is required for the XDP offload to work correctly. We use the "QM" AF_Packet mode which binds Suricata threads to driver queues, and it is card's responsibility to hash flows symmetrically between those queues.

Instructions how to configure symmetric RSS are further down this article.

For generic information how to configure your system firmware (BIOS/UEFI settings are important) and how Linux data processing works and how to measure packet loss, see SEPTun Mark I. The only thing that changes here is the symmetric RSS. We will recommend the same kernel and driver settings and pin Suricata workers to cores and move all workload that can be moved off worker's cores.

Packages

Specific to our setup of Suricata features:

```
sudo apt-get -y install git build-essential autoconf automake \
libtool pkg-config  libpcre3 libpcre3-dbg libpcre3-dev \
libpcap-dev libnet1-dev  libyaml-0-2 libyaml-dev zlib1g \
zlib1g-dev  libmagic-dev libcap-ng-dev libjansson-dev \
libjansson4 libnss3-dev libnspr4-dev libgeoip-dev libluajit-5.1-dev \
rustc cargo
```

Clang & elf

Make sure you have clang & elf installed on the system for XDP. The default clang version should do.

```
apt-get install clang libelf-dev
```

Kernel and NIC related

- Newer kernel that supports XDP (4.13.10+ in our case)
- Depending on the af-packet mode (described later in the article) - RSS symmetric hashing on the NIC (Intel 82599ES 10-Gigabit/x520/x540 in our case)
- In tree kernel drivers NIC drivers. (aka downloading and compiling your drivers did not seem to work - explained how to do it further down in the article)

If you need help, you can build the desired kernel version with the help of the scripts here

Disable irqbalance

```
systemctl stop irqbalance
systemctl disable irqbalance
```

(make sure it is gone :) from the system)

BPF

A patched BPF (headers) are also needed:

```
git clone -b libbpf-release  https://github.com/regit/linux.git
cd linux/tools/lib/bpf/
make clean && make
sudo make install && sudo make install_headers
sudo ldconfig
```

Compile and install Suricata

The Suricata config/compile below includes some extra functionality that we needed (like Rust/file extraction/geoip etc..) to be build. The one important for enabling XDP functionality is `--enable-ebpf --enable-ebpf-build` in the config line. Below we use Suricata git master but the XDP functionality is present in Suricata 4.1+

```
git clone  https://github.com/OISF/suricata.git
cd suricata && \
git clone https://github.com/OISF/libhtp.git -b 0.5.x

./autogen.sh

CC=clang-4.0 ./configure \
--prefix=/usr/ --sysconfdir=/etc/ --localstatedir=/var/ \
--with-libnss-libraries=/usr/lib \
--with-libnss-includes=/usr/include/nss/ \
--with-libnspr-libraries=/usr/lib \
--with-libnspr-includes=/usr/include/nspr \
--enable-geoip --enable-luajit --enable-rust \
--enable-ebpf --enable-ebpf-build

make clean && make
sudo  make install-full
sudo ldconfig
```

Copy the resulting xdp filter as needed - you can specify a particular path in `suricata.yaml`. In our case it wwas `/etc/suricata/`:

```
cp ebpf/xdp_filter.bpf /etc/suricata/
```

Setup af-packet section/interface in `suricata.yaml`. We will use `cluster_qm` as we can have symmetric hashing on the NIC, `xdp-mode:  driver` and we will also use the `/etc/suricata/xdp_filter.bpf` (in our example TCP offloading/bypass)

Using one interface in the suricata.yaml config -

```
- interface: eth3
  threads: 14
  cluster-id: 97
  cluster-type: cluster_qm # symmetric hashing  is a must!
  defrag: yes
  # eBPF file containing a 'loadbalancer' function that will be inserted
  # into the  kernel and used as load balancing function
  #ebpf-lb-file:  /etc/suricata/lb.bpf
  # eBPF file containing a 'filter' function that will be inserted into
  # the kernel and used as packet filter function
```

```
# eBPF file containing a 'xdp' function that will be inserted into the
# kernel and used as XDP packet filter function
#ebpf-filter-file:  /etc/suricata/filter.bpf
# Xdp mode, "soft" for skb based version, "driver" for network card
# based and "hw" for card supporting eBPF.
xdp-mode: driver
xdp-filter-file:  /etc/suricata/xdp_filter.bpf
# if the ebpf filter implements a bypass function, you can set
# 'bypass' to yes and benefit from these feature
bypass: yes
use-mmap: yes
mmap-locked: yes
# Use tpacket_v3, capture mode, only active if user-mmap is true
tpacket-v3: yes
ring-size: 200000
block-size: 1048576
```

Using two interface in the suricata.yaml config -

```
- interface: eth3
  threads: 7
  cluster-id: 97
  cluster-type: cluster_qm
  defrag: yes
  # eBPF file containing a 'loadbalancer' function that will be inserted
  # into the kernel and used as load balancing function
  #ebpf-lb-file:  /etc/suricata/lb.bpf
  # eBPF file containing a 'filter' function that will be inserted into
  # the kernel and used as packet filter function
  # eBPF file containing a 'xdp' function that will be inserted into the
  # kernel and used as XDP packet filter function
  #ebpf-filter-file:  /etc/suricata/filter.bpf
  # Xdp mode, "soft" for skb based version, "driver" for network card
  # based and "hw" for card supporting eBPF.
  xdp-mode: driver
  xdp-filter-file:  /etc/suricata/xdp_filter.bpf
  # if the ebpf filter implements a bypass function, you can set
  # 'bypass' to yes and benefit from these feature
  bypass: yes
  use-mmap: yes
  mmap-locked: yes
  # Use tpacket_v3, capture mode, only active if user-mmap is true
  tpacket-v3: yes
  ring-size: 200000
  block-size: 1048576

- interface: eth2
```

```
  threads: 7
  cluster-id: 98
  cluster-type: cluster_qm
  defrag: yes
  # eBPF file containing a 'loadbalancer' function that will be inserted
  # into the kernel and used as load balancing function
  #ebpf-lb-file:  /etc/suricata/lb.bpf
  # eBPF file containing a 'filter' function that will be inserted into
  # the kernel and used as packet filter function
  # eBPF file containing a 'xdp' function that will be inserted into the
  # kernel and used as XDP packet filter function
  #ebpf-filter-file:  /etc/suricata/filter.bpf
  # Xdp mode, "soft" for skb based version, "driver" for network card
  # based and "hw" for card supporting eBPF.
  xdp-mode: driver
  xdp-filter-file:  /etc/suricata/xdp_filter.bpf
  # if the ebpf filter implements a bypass function, you can set
  # 'bypass' to yes and benefit from these feature
  bypass: yes
  use-mmap: yes
  mmap-locked: yes
  # Use tpacket_v3, capture mode, only active if user-mmap is true
  tpacket-v3: yes
  ring-size: 200000
  block-size: 1048576
```

Also enable "bypass" in the "stream" section:

```
stream:
  bypass: true
```

An example of one of the test machines set up also includes (example of a `stream.reassembly.deprh`):

```
stream:
 memcap: 14gb
 checksum-validation: no
 bypass: yes
 prealloc-sessions: 375000
 inline: auto
 reassembly:
   memcap: 20gb
   depth: 1mb
   toserver-chunk-size: 2560
   toclient-chunk-size: 2560
   randomize-chunk-size: yes
   randomize-chunk-range: 10
   raw: yes
   segment-prealloc: 200000
```

and some timeouts like these:

```
flow-timeouts:

 default:
   new: 3
   established: 30
   closed: 0
   bypassed: 20
   emergency-new: 1
   emergency-established: 10
   emergency-closed: 0
   emergency-bypassed: 5
 tcp:
   new: 3
   established: 30
   closed: 1
   bypassed: 25
   emergency-new: 1
   emergency-established: 10
   emergency-closed: 0
   emergency-bypassed: 5
 udp:
   new: 3
   established: 30
   bypassed: 20
   emergency-new: 1
   emergency-established: 10
   emergency-bypassed: 5
 icmp:
   new: 2
   established: 30
   bypassed: 20
   emergency-new: 1
   emergency-established: 10
   emergency-bypassed: 5
```

Setup symmetric hashing on the NIC

The Why

RSS is technology initially defined by Microsoft. It does a general load balancing of network data over multiple cores or CPUs by using IP tuple to calculate a hash value.

The problem from the IDS/IPS perspective is that it needs to see the traffic as the end client will to do its job correctly. The challenge with RSS is that it RSS has been

made for another purpose for example scaling of large web/filesharing installations and thus not needing the same "flow" consistency as an IDS/IPS deployment.

As explained clearly in the Suricata documentation:

" *Receive Side Scaling is a technique used by network cards to distribute incoming traffic over various queues on the NIC. This is meant to improve performance, but it is important to realize that it was designed for average traffic, not for the IDS packet capture scenario. RSS using a hash algorithm to distribute the incoming traffic over the various queues. This hash is normally not symmetrical. This means that when receiving both sides of flow, each side may end up in a different queue. Sadly, when deploying Suricata, this is the typical scenario when using span ports or taps.* "

In other words in the majority of RSS cases - the hash of

- `ipsrc=1.1.1.1,ipdst=2.2.2.2,sport=11111,dport=22222`

is NOT the same as the hash of

- `ipsrc=2.2.2.2,ipdst=1.1.1.1,sport=22222,dport=11111`

....so not going to the same queue/thread!

It turns out that on specific Intel NICs you can enable symmetric RSS with a low entropy key. In that case - we could use AF_PACKET with `cluster_qm` - all packets linked by network card to a RSS queue are sent to the same socket. This requires at least Linux 3.14. (FYI - other methods are also available)

Follow these instructions closely for desired result (per interface -in this case `eth3`):

```
ifconfig eth3 down
```

Use and load in tree kernel drivers for the NIC **NOTE:** In this case the kernel and sources used is 4.15.2

```
cd /lib/modules/4.15.2-amd64/kernel/drivers/net/ethernet/intel/ixgbe
rmmod ixgbe && insmod ixgbe.ko MQ=1,1 RSS=0,0 \
InterruptThrottleRate=12500,12500 LRO=0,0 vxlan_rx=0,0
```

Enable symmetric hashing and set queues on the NIC

ifconfig eth3 down
ethtool -L eth3 combined 14
ethtool -K eth3 rxhash on
ethtool -K eth3 ntuple on
ifconfig eth3 up

./set_irq_affinity 2-15 eth3
ethtool -X eth3 hkey 6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A:6D:5A equal 14
ethtool -x eth3
ethtool -n eth3

In the above set up you are free to use any recent `set_irq_affinity` script. It is available in any Intel x520/710 NIC sources driver download.

Keep in mind that the number of `combined` must match what's later used for set_irq_affinity so if you have 16 queues, pin workers to 16 threads.

We would recommend saving 1-2 cores for each NUMA node for OS and Suricata threads that do not do packet processing.

NOTE: We use a particular low entropy key for the symmetric hashing. More info about the research for symmetric hashing set up

AMD

For AMD CPUs system it is recommended (based on our tests) that the general set up above is followed except for dedicating/pining the Suricata worker threads on different NUMA node CPUs (`lscpu` will show you which ones are those) than the one for the NIC (which is the opposite we do with Intel CPU based systems). This will leverage the AMD Hypertransport technology for better performance.

For example:

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                48
On-line CPU(s) list:   0-47
Thread(s) per core:    2
Core(s) per socket:    24
Socket(s):             1
NUMA node(s):          4
Vendor ID:             AuthenticAMD
CPU family:            23
Model:                 1
Model name:            AMD EPYC 7401 24-Core Processor
Stepping:              2
CPU MHz:               2000.000
CPU max MHz:           2000.0000
CPU min MHz:           1200.0000
BogoMIPS:              3991.97
Virtualization:        AMD-V
L1d cache:             32K
```

```
L1i cache:             64K
L2 cache:              512K
L3 cache:              8192K
NUMA node0 CPU(s):     0-5,24-29
NUMA node1 CPU(s):     6-11,30-35
NUMA node2 CPU(s):     12-17,36-41
NUMA node3 CPU(s):     18-23,42-47
```

So in the example above if your NIC is on `NUMA 0` you would do something like:

- enable 6 (or any number up to 12 RSS - you need to experiment to see what is best for your set up)
- pin those interrupts to the CPUs on `NUMA 0` (same as the card)
- use cpu affinity with Suricata (`suricata.yaml`) and make sure the af-packet worker threads are running on anything but `NUMA 0`
- use `cluster_flow` (`af-packet` config section in `suricata.yaml`)

Disable the NIC offloading

```
for i in rx tx tso ufo gso gro lro tx nocache copy sg txvlan rxvlan; do
       /sbin/ethtool -K eth3 $i off 2>&1 > /dev/null;
done
```

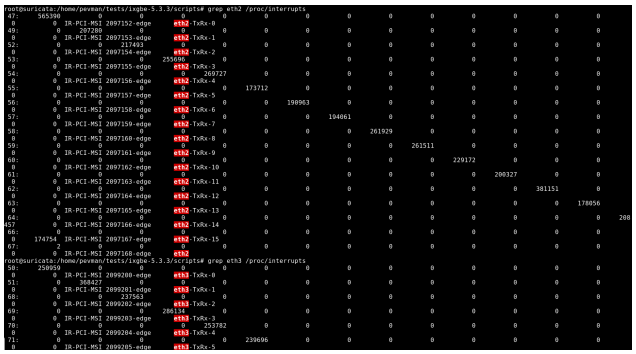
Balance as much as you can

Try to use the network's card balancing as much as possible(in a script for example you do/add in):

```
for proto in tcp4 udp4 ah4 esp4 sctp4 tcp6 udp6 ah6 esp6 sctp6; do
       /sbin/ethtool -N eth3 rx-flow-hash $proto sdfn
done
```

IRQ affinity

Make sure you have the irq affinity correct. Example



Check with (example) `grep eth2 /proc/interrupts`

Start Suricata with XDP

Make sure you have the stats enabled in section `eve.json` in `suricata.yaml`:

```
- stats:
    totals: yes       # stats for all threads merged together
    threads: no       # per thread stats
    deltas: yes        # include delta values
```

```
/usr/bin/suricata -c /etc/suricata/xdp-suricata.yaml \
--pidfile /var/run/suricata.pid  --af-packet=eth3 -vvv
```

Confirm you have the XDP filter engaged in the output (example)

```
(runmode-af-packet.c:220) <Config> (ParseAFPConfig) \
-- Enabling locked memory for mmap on iface eth3
(runmode-af-packet.c:231) <Config> (ParseAFPConfig) \
-- Enabling tpacket v3 capture on iface eth3
(runmode-af-packet.c:326) <Config> (ParseAFPConfig) \
-- Using queue based cluster mode for AF_PACKET (iface eth3)
(runmode-af-packet.c:424) <Info> (ParseAFPConfig) \
-- af-packet will use '/etc/suricata/xdp_filter.bpf' as XDP filter file
(runmode-af-packet.c:429) <Config> (ParseAFPConfig) \
-- Using bypass kernel functionality for AF_PACKET (iface eth3)
(runmode-af-packet.c:609) <Config> (ParseAFPConfig) \
-- eth3: enabling zero copy mode by using data release call
(util-runmodes.c:296) <Info> (RunModeSetLiveCaptureWorkersForDevice) \
-- Going to use 8 thread(s)
```

Have a look at the stats to see how are you doing:

```
sudo tail -F /var/log/suricata/eve.json  |grep stats \
|jq 'select(.event_type=="stats")\
|{bypassed: .stats.flow_bypassed, bytes:.stats.decoder.bytes, \
bytes_delta: .stats.decoder.bytes_delta,percent: \
(.stats.flow_bypassed.bytes / .stats.decoder.bytes * 100)}' -
```

You can/should see visible difference right away (for example in one of our test runs about `10%`):


```
{
 "bypassed": {
   "closed": 3608681,
   "closed_delta": 6734,
   "pkts": 537177957,
   "pkts_delta": 604972,
   "bytes": 575615096046,
   "bytes_delta": 526741771
 },
 "bytes": 5278384812258,
 "bytes_delta": 3717316268,
 "percent": 10.905137016711027
}
{
 "bypassed": {
   "closed": 3614378,
   "closed_delta": 5697,
   "pkts": 537888772,
   "pkts_delta": 710815,
   "bytes": 576283121574,
   "bytes_delta": 668025528
 },
 "bytes": 5281159632142,
 "bytes_delta": 2774819884,
 "percent": 10.912056474616803
}
```


Of course depending on the type of traffic - the percentage could be much more.

Pros

- XDP provides a serious performance boost for native Linux drivers by introducing the `XDP bypass` functionality to Suricata which in turn allows for dealing with elephant flows much earlier in the critical packet path. Thus offloading significant work from Suricata and the kernel regarding the capability to bypass flows before Suricata process them - minimizing the perf intensive work needed to be done.

Caveats

- the current Suricata XDP eBPF implementation allows for TCP only flow bypass
- other eBPF filters can be done though it requires you to write your filters

Bugs and info

During the testing and research, there were a few bugs/optimizations discovered and patches submitted by Eric Leblond(@regiteric) regarding the Linux kernel and Peter Manev(@pevma) concerning Intel NIC. Some further patches (and testing) that helped were introduced by Jesper Brouer (@netoptimiser) as well.

Bingo bug

Intel NIC interrupts.

One of the most critical bugs as of the moment of writing this article is the IRQs reset one (on some kernel version/ Intel NIC combo). It seems right after Suricata starts and the eBPF script code gets injected all interrupts are pinned to CPU0 (and we need them to be spread - not like that.)

NOTE: In our tests (with the NICs specified at the beginning of the article) it showed it affected kernel `4.13.10` and possibly `4.14.x/4.15RC`. With kernel `4.15.x(4.15.2)` we had no problem.

You can quickly check if the interrupts are spread correctly (or pinned to CPU0) after Suricata starts with - `grep eth2 /proc/interrupts` (see section `IRQ affinity` above). It is recommended you do that check on a per kernel/NIC driver upgrade.

Current bingo fix

The fix (if you are not on 4.15.2 and above) would be to remap the interrupts right after Suricata starts (using `set_irq_affinity` script as explained above). As of the moment there is no definitive solution devised yet by Intel.

XDP bypass with cpumap to the rescue (if needed)

To counter the problem above (and the case of no symmetric RSS availability) in a more permanent basis - the `xdp_cpumap` concept code was ported and introduced into Suricata by Eric Leblond in January 2018. The `xdp_cpumap` is a recent Linux kernel xdp feature that (among other things) enables redirection XDP frames to remote CPUs (page 13) without breaking the flow.

How to use it

In case of no RSS symmetric hashing available you can try `xdp_cpumap` Make sure you are sitting on Linux kernel 4.15.+

To make use of the `xdp_cpumap` functionality, you would need to:

- use xdp-cpu-redirect: [xx-xx] (af_packet section of suriata.yaml)
- use `cluster-type: cluster_cpu` (af_packet section of suriata.yaml)
- make sure you pin all interrupts of the NIC to one CPU (make sure it is on the same NUMA as on the NIC)
- have CPU affinity enabled and used on the cores of the appropriate NUMA node wise for the NIC location cores.

Example:

```
# Suricata is multi-threaded. Here the threading can be influenced.
threading:
 set-cpu-affinity: yes
 # Tune cpu affinity of threads. Each family of threads can be bound
 # on specific CPUs.
 #
 # These 2 apply to the all runmodes:
 # management-cpu-set is used for flow timeout handling, counters
 # worker-cpu-set is used for 'worker' threads
 #
 # Additionally, for autofp these apply:
 # receive-cpu-set is used for capture threads
 # verdict-cpu-set is used for IPS verdict threads
 #
 cpu-affinity:
   - management-cpu-set:
       cpu: [ "all" ]  # include only these cpus in affinity settings
       prio:
         default: "low"
   - receive-cpu-set:
       cpu: [ 2-11 ]  # include only these cpus in affinity settings
   - worker-cpu-set:
       #
       cpu: [ "2-11" ]
       mode: "exclusive"
       # Use explicitely 3 threads and don't compute number by using
       # detect-thread-ratio variable:
       # threads: 3
       prio:
         default: "high"
```

```
#- verdict-cpu-set:
#    cpu: [ 0 ]
#    prio:
#      default: "high"
```

Tools

Some nifty tools used during the research:

- nstat (thanks to @netoptimiser)

```
nstat > /devnstat > /dev/null && sleep 1 && nstat
```

- mpstat

```
mpstat -u -I ALL -P ALL
```

- smp_affinity_list

```
grep -H . /proc/irq/*/eth*/../smp_affinity_list
```

- top

```
top -H -p `pidof suricata`
```

- very useful NIC stats tool (thanks to @netoptimiser again)

https://github.com/netoptimizer/network-testing/blob/master/bin/ethtool_stats.pl

- pidstat (thanks to @netoptimiser again)

```
pidstat -u -t 2
pidstat -w 2
```

- strace

```
strace -c your_comand_here
```

- perf top/stats - what would you do without those :)

```
perf stat -e 'syscalls:sys_enter_*' \
-p 2894 -a sleep 7200 &> suri.syscalls.logs

perf top -C 0
....
```

Chapter 3

The Meltdown and Spectre

As we were finalizing our tests and experiments Suricata and XDP bypass - the big Meltdown happened.

The security vulnerability was patched, but there is widespread expectation that the patch itself is going to hurt the syscall performance on any Linux system. The performance hit expectation is between 5-25% depending on the specific application usage. We were curious about what the impact on those might be regarding Suricata and the latest XDP code additions - hence we did some measurements.

In the case of Suricata there are two types of measurements that we did:

- running on live traffic
- reading pcaps

The first challenge with measurements while Suricata is running live is that its performance depends on that actual amount and type of traffic - which has some expected deviations even if you make consecutive/repetitive measurements.

The second challenge is that the measurement tools themselves introduce performance overhead so we can not exclusively concentrate on performance only - but rather to see if there is any apparent and appalling deviation in terms of syscalls.

We did some benchmarking that is documented below.

NOTE: Suricata used to do the test run was compiled with debugging symbols enabled. That plus the usage of `strace` can have performance impact alone. So we are not interested in how fast it completed but rather than if there is any bigger deviations between the patched and unpatched kernels in terms of syscalls (duration/occurrence).

Strace overhead is explained here (section - perf vs strace)

Suricata version used in both pre and post patched kernel:

```
root@suricata:/home/pevman/tests/kernel# suricata --build-info
This is Suricata version 4.1.0-dev (rev f815027)
Features: PCAP_SET_BUFF PF_RING AF_PACKET HAVE_PACKET_FANOUT
LIBCAP_NG LIBNET1.1 HAVE_HTP_URI_NORMALIZE_HOOK PCRE_JIT HAVE_NSS
HAVE_LUA HAVE_LUAJIT HAVE_LIBJANSSON TLS MAGIC RUST
```

```
SIMD support: SSE_4_2 SSE_4_1 SSE_3
Atomic intrisics: 1 2 4 8 16 byte(s)
64-bits, Little-endian architecture
GCC version 4.2.1 Compatible Clang 3.8.0 (tags/RELEASE_380/final),
C version 199901
compiled with _FORTIFY_SOURCE=0
L1 cache line size (CLS)=64
thread local storage method: __thread
compiled with LibHTP v0.5.25, linked against LibHTP v0.5.25

Suricata Configuration:
 AF_PACKET support:                       yes
 eBPF support:                            yes
 XDP support:                             yes
 PF_RING support:                         yes
 NFQueue support:                         no
 NFLOG support:                           no
 IPFW support:                            no
 Netmap support:                          no
 DAG enabled:                             no
 Napatech enabled:                        no

 Unix socket enabled:                     yes
 Detection enabled:                       yes

 Libmagic support:                        yes
 libnss support:                          yes
 libnspr support:                         yes
 libjansson support:                      yes
 liblzma support:                         yes
 hiredis support:                         no
 hiredis async with libevent:             no
 Prelude support:                         no
 PCRE jit:                                yes
 LUA support:                             yes, through luajit
 libluajit:                               yes
 libgeoip:                                yes
 Non-bundled htp:                         no
 Old barnyard2 support:                   no
 Hyperscan support:                       yes
 Libnet support:                          yes

 Rust support (experimental):             yes
 Experimental Rust parsers:               no
 Rust strict mode:                        yes
 Rust debug mode:                         no
```

```
 Suricatasc install:                    yes

 Profiling enabled:                     no
 Profiling locks enabled:               no

Development settings:
 Coccinelle / spatch:                   yes
 Unit tests enabled:                    no
 Debug output enabled:                  no
 Debug validation enabled:              no

Generic build parameters:
 Installation prefix:                   /usr/local
 Configuration directory:               /usr/local/etc/suricata/
 Log directory:                         /usr/local/var/log/suricata/

 --prefix                               /usr/local
 --sysconfdir                           /usr/local/etc
 --localstatedir                        /usr/local/var

 Host:                                  x86_64-pc-linux-gnu
 Compiler:                              clang-3.8 (exec name) / clang (rea
 GCC Protect enabled:                   no
 GCC march native enabled:              yes
 GCC Profile enabled:                   no
 Position Independent Executable enabled: no
 CFLAGS                                 -ggdb -O0 -march=native
 -I${srcdir}/../rust/gen/c-headers
 PCAP_CFLAGS                             -I/usr/include
 SECCFLAGS
```

Unpatched kernel - reading pcaps

root@suricata:/home/pevman/tests/kernel# uname -a
Linux suricata 4.13.10-amd64 #1 SMP Mon Oct 30 23:01:00 CET 2017 x86_64 x86_64 x86_64 GNU/Linux

pcap run 1

Using:

- 160GB pcap live ISP capture
- 30k ETPro rules

Command:

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k none \
-r /var/log/suricata/pcap/bigtest.pcap --runmode=autofp \
&> strace-bigpcap-run-1
```

Output:

```
[12087] 9/2/2018 -- 00:53:10 - (conf-yaml-loader.c:265) <Info> (ConfYamlPars
[12087] 9/2/2018 -- 00:53:10 - (suricata.c:1073) <Notice> (LogVersion) -- Th
[12087] 9/2/2018 -- 00:54:04 - (tm-threads.c:2172) <Notice> (TmThreadWaitOnT
[12087] 9/2/2018 -- 01:21:25 - (suricata.c:2716) <Notice> (SuricataMainLoop)
[12102] 9/2/2018 -- 01:21:33 - (source-pcap-file.c:354) <Notice> (ReceivePca
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
99.92  699.882916        3433    203893           nanosleep
 0.07    0.519546           4    140138       158 futex
 0.00    0.011924          23       512           munmap
 0.00    0.006752         614        11           madvise
 0.00    0.005745           1      5423           brk
 0.00    0.000887           2       589           mmap
 0.00    0.000765           0      4865         2 read
 0.00    0.000456           5       100           mprotect
 0.00    0.000320           5        69           open
 0.00    0.000225           7        34        34 access
 0.00    0.000194           3        69           close
 0.00    0.000169           3        60           fstat
 0.00    0.000150           5        32           clone
 0.00    0.000047           1        74           write
 0.00    0.000005           5         1           arch_prctl
 0.00    0.000003           3         1           bind
 0.00    0.000003           2         2           unlink
 0.00    0.000001           0        22         2 stat
 0.00    0.000000           0         3           lseek
 0.00    0.000000           0        10           rt_sigaction
 0.00    0.000000           0         3           rt_sigprocmask
 0.00    0.000000           0         2         2 ioctl
 0.00    0.000000           0         1           socket
 0.00    0.000000           0         1           listen
 0.00    0.000000           0         1           setsockopt
 0.00    0.000000           0         1           execve
 0.00    0.000000           0         1           uname
 0.00    0.000000           0         6           getdents
 0.00    0.000000           0        14        14 mkdir
 0.00    0.000000           0         1           chmod
 0.00    0.000000           0         2           getrlimit
```

```
 0.00    0.000000           0         2           sysinfo
 0.00    0.000000           0         3           prctl
 0.00    0.000000           0         1           setrlimit
 0.00    0.000000           0        74           gettid
 0.00    0.000000           0         1           set_tid_address
 0.00    0.000000           0         1           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00  700.430108              356023       212 total
```

pcap run 2

Using:

- 160GB pcap live ISP capture
- 30k ETPro rules

NOTE: This is the same run as above. The purpose of this second identical run is to show us if there are some small deviations and where to expect those so that we can have that in mind when we do the runs over the patched kernel.

Command:

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k none \
-r /var/log/suricata/pcap/bigtest.pcap --runmode=autofp \
&> strace-bigpcap-run-2
```

Output:

```
[13707] 9/2/2018 -- 01:21:35 - (conf-yaml-loader.c:265) <Info> (ConfYamlPars
[13707] 9/2/2018 -- 01:21:35 - (suricata.c:1073) <Notice> (LogVersion) -- Th
[13707] 9/2/2018 -- 01:22:21 - (tm-threads.c:2172) <Notice> (TmThreadWaitOnT
[13707] 9/2/2018 -- 01:50:03 - (suricata.c:2716) <Notice> (SuricataMainLoop)
[13761] 9/2/2018 -- 01:50:10 - (source-pcap-file.c:354) <Notice> (ReceivePca
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
99.92  646.074279        3032    213119           nanosleep
 0.07    0.477696           3    139208       203 futex
 0.00    0.014877          30       495           munmap
 0.00    0.008516           2      5423           brk
 0.00    0.003147         629         5           madvise
 0.00    0.001597           3       589           mmap
 0.00    0.001145          17        69           open
 0.00    0.001144           0      4865         2 read
 0.00    0.000589           6       100           mprotect
 0.00    0.000273           5        60           fstat
 0.00    0.000252           3        74           write
 0.00    0.000249           4        69           close
```

```
 0.00    0.000230           7        34        34 access
 0.00    0.000177           6        32           clone
 0.00    0.000085           4        22         2 stat
 0.00    0.000068           5        14        14 mkdir
 0.00    0.000042           7         6           getdents
 0.00    0.000028           3        10           rt_sigaction
 0.00    0.000022           0        74           gettid
 0.00    0.000013           4         3           rt_sigprocmask
 0.00    0.000012           4         3           lseek
 0.00    0.000012           6         2           getrlimit
 0.00    0.000009           5         2         2 ioctl
 0.00    0.000007           7         1           execve
 0.00    0.000005           3         2           sysinfo
 0.00    0.000005           2         3           prctl
 0.00    0.000004           2         2           unlink
 0.00    0.000004           4         1           arch_prctl
 0.00    0.000004           4         1           set_tid_address
 0.00    0.000004           4         1           set_robust_list
 0.00    0.000002           2         1           socket
 0.00    0.000002           2         1           bind
 0.00    0.000002           2         1           uname
 0.00    0.000001           1         1           setsockopt
 0.00    0.000001           1         1           chmod
 0.00    0.000000           0         1           listen
 0.00    0.000000           0         1           setrlimit
------ ----------- ----------- --------- --------- ----------------
100.00  646.584503                364296       257 total
```

pcap run 3

Using:

- 8.1GB pcap live ISP capture
- 30k ETPro rules

NOTE: This is the same run as above but on a different size pcap. The purpose of this second run is to show us if there are some small deviations and where to expect those so that we can have that in mind when we do the runs over the patched kernel.

Command:

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k none \
-r /var/log/suricata/pcap/test.pcap --runmode=autofp \
&> strace-smallpcap-run-1
```

Output:

```
[11774] 9/2/2018 -- 00:49:16 - (conf-yaml-loader.c:265) <Info> (ConfYamlPars
[11774] 9/2/2018 -- 00:49:16 - (suricata.c:1073) <Notice> (LogVersion) -- Th
[11774] 9/2/2018 -- 00:50:10 - (tm-threads.c:2172) <Notice> (TmThreadWaitOnT
[11774] 9/2/2018 -- 00:51:21 - (suricata.c:2716) <Notice> (SuricataMainLoop)
[11836] 9/2/2018 -- 00:51:25 - (source-pcap-file.c:354) <Notice> (ReceivePca
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
98.10   33.157150         836     39663           nanosleep
 1.13    0.381366           5     78731        51 futex
 0.63    0.212663        9667        22         2 stat
 0.06    0.020189           3      6122           brk
 0.02    0.007494        3747         2           unlink
 0.02    0.006948          14       500           munmap
 0.02    0.006781           1      4865         2 read
 0.01    0.002132         355         6           madvise
 0.00    0.001489           3       589           mmap
 0.00    0.000943          14        69           open
 0.00    0.000427          13        32           clone
 0.00    0.000333           3       100           mprotect
 0.00    0.000220           4        60           fstat
 0.00    0.000158           2        69           close
 0.00    0.000142           4        34        34 access
 0.00    0.000110           1        74           write
 0.00    0.000077           1        74           gettid
 0.00    0.000021          21         1           setsockopt
 0.00    0.000017           2        10           rt_sigaction
 0.00    0.000013          13         1           chmod
 0.00    0.000011           2         6           getdents
 0.00    0.000010           5         2         2 ioctl
 0.00    0.000009           9         1           bind
 0.00    0.000006           2         3           prctl
 0.00    0.000003           1         3           lseek
 0.00    0.000003           1         3           rt_sigprocmask
 0.00    0.000002           2         1           socket
 0.00    0.000002           2         1           listen
 0.00    0.000002           2         1           arch_prctl
 0.00    0.000001           1         2           getrlimit
 0.00    0.000001           1         1           set_tid_address
 0.00    0.000001           1         1           set_robust_list
 0.00    0.000000           0         1           execve
 0.00    0.000000           0         1           uname
 0.00    0.000000           0        14        14 mkdir
 0.00    0.000000           0         2           sysinfo
 0.00    0.000000           0         1           setrlimit
------ ----------- ----------- --------- --------- ----------------
100.00   33.798724              131068       105 total
```

pcap run 4

Using:

- 8.1GB pcap live ISP capture
- 30k ETPro rules

NOTE: This is the same run as above. The purpose of this second identical run is to show us if there are some small deviations and where to expect those so that we can have that in mind when we do the runs over the patched kernel.

Command:

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k none \
-r /var/log/suricata/pcap/test.pcap --runmode=autofp \
&> strace-smallpcap-run-2
```

Output:

```
[11936] 9/2/2018 -- 00:51:27 - (conf-yaml-loader.c:265) <Info> (ConfYamlPars
[11936] 9/2/2018 -- 00:51:27 - (suricata.c:1073) <Notice> (LogVersion) -- Th
[11936] 9/2/2018 -- 00:52:17 - (tm-threads.c:2172) <Notice> (TmThreadWaitOnT
[11936] 9/2/2018 -- 00:53:04 - (suricata.c:2716) <Notice> (SuricataMainLoop)
[11990] 9/2/2018 -- 00:53:08 - (source-pcap-file.c:354) <Notice> (ReceivePca

% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
98.75   28.964359         823     35195           nanosleep
 1.15    0.336672           4     79472        76 futex
 0.05    0.013790           2      6011           brk
 0.02    0.007125          14       507           munmap
 0.02    0.005256         876         6           madvise
 0.00    0.001043           2       589           mmap
 0.00    0.000760           0      4865         2 read
 0.00    0.000349           5        69           open
 0.00    0.000342           3       100           mprotect
 0.00    0.000211           7        32           clone
 0.00    0.000173           3        69           close
 0.00    0.000142           2        60           fstat
 0.00    0.000141           4        34        34 access
 0.00    0.000066           5        14        14 mkdir
 0.00    0.000058           1        74           write
 0.00    0.000037           2        22         2 stat
 0.00    0.000015           0        74           gettid
 0.00    0.000011           2         6           getdents
 0.00    0.000011           6         2           sysinfo
 0.00    0.000008           3         3           lseek
```

```
 0.00    0.000006           2         3           prctl
 0.00    0.000005           5         1           arch_prctl
 0.00    0.000003           3         1           uname
 0.00    0.000003           2         2           getrlimit
 0.00    0.000003           3         1           setrlimit
 0.00    0.000002           1         2         2 ioctl
 0.00    0.000000           0        10           rt_sigaction
 0.00    0.000000           0         3           rt_sigprocmask
 0.00    0.000000           0         1           socket
 0.00    0.000000           0         1           bind
 0.00    0.000000           0         1           listen
 0.00    0.000000           0         1           setsockopt
 0.00    0.000000           0         1           execve
 0.00    0.000000           0         2           unlink
 0.00    0.000000           0         1           chmod
 0.00    0.000000           0         1           set_tid_address
 0.00    0.000000           0         1           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00   29.330591              127237       130 total
```

Unpatched kernel - running live

Using: - 30k ETPro rules

We measured Suricata running live process for 2 hours 3 different sample times during a 24 hr period.

Command:

```
perf stat -e 'syscalls:sys_enter_*' \
-p 2894 -a sleep 7200 &> suri.syscalls-1
```

Output:

```
root@suricata:/home/pevman/tests/meltdown/prepatched-kernel#
cat suri.syscalls-1 |grep -v ' 0 '|grep syscalls | sort -rn
      151,125,258      syscalls:sys_enter_write
       67,712,628      syscalls:sys_enter_futex
       51,441,437      syscalls:sys_enter_poll
          709,513      syscalls:sys_enter_nanosleep
          118,331      syscalls:sys_enter_getsockopt
           35,813      syscalls:sys_enter_select
           12,595      syscalls:sys_enter_mprotect
              905      syscalls:sys_enter_munmap
              457      syscalls:sys_enter_mmap
               42      syscalls:sys_enter_madvise
               12      syscalls:sys_enter_mkdir
                2      syscalls:sys_enter_open
```

```
                2      syscalls:sys_enter_newfstat
                2      syscalls:sys_enter_lseek
                2      syscalls:sys_enter_close

root@suricata:/home/pevman/tests/meltdown/prepatched-kernel#
cat suri.syscalls-2 |grep -v ' 0 '|grep syscalls | sort -rn
      118,638,811      syscalls:sys_enter_write
       75,667,793      syscalls:sys_enter_futex
       33,108,479      syscalls:sys_enter_poll
          707,233      syscalls:sys_enter_nanosleep
          246,737      syscalls:sys_enter_mprotect
           74,174      syscalls:sys_enter_getsockopt
           35,810      syscalls:sys_enter_select
            6,272      syscalls:sys_enter_munmap
            3,298      syscalls:sys_enter_mmap
              119      syscalls:sys_enter_madvise
               92      syscalls:sys_enter_brk
               12      syscalls:sys_enter_mkdir
                4      syscalls:sys_enter_gettid
                2      syscalls:sys_enter_open
                2      syscalls:sys_enter_newfstat
                2      syscalls:sys_enter_lseek
                2      syscalls:sys_enter_getrandom
                2      syscalls:sys_enter_close

root@suricata:/home/pevman/tests/meltdown/prepatched-kernel#
cat suri.syscalls-3 |grep -v ' 0 '|grep syscalls | sort -rn
       64,659,097      syscalls:sys_enter_write
       53,363,829      syscalls:sys_enter_poll
       42,849,286      syscalls:sys_enter_futex
          709,535      syscalls:sys_enter_nanosleep
          120,551      syscalls:sys_enter_getsockopt
           35,814      syscalls:sys_enter_select
            2,676      syscalls:sys_enter_munmap
            1,472      syscalls:sys_enter_mprotect
            1,344      syscalls:sys_enter_mmap
               52      syscalls:sys_enter_madvise
               12      syscalls:sys_enter_mkdir
                2      syscalls:sys_enter_open
                2      syscalls:sys_enter_newfstat
                2      syscalls:sys_enter_lseek
                2      syscalls:sys_enter_getrandom
                2      syscalls:sys_enter_close
```

Patched kernel - reading pcaps

root@suricata:~# uname -a
Linux suricata 4.15.2-amd64 #1 SMP Thu Feb 8 23:36:33 CET 2018 x86_64 x86_64 x86_64 GNU/Linux

pcap run 1

Using:

- 160GB pcap live ISP capture
- 30k ETPro rules

NOTE: This is the same run as above. The purpose of this second identical run is to show us if there are some small deviations and where to expect those so that we can have that in mind when we do the runs over the patched kernel.

Command:

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k none \
-r /var/log/suricata/pcap/bigtest.pcap --runmode=autofp \
&> strace-bigpcap-run-2
```

Output:

```
[5053] 9/2/2018 -- 11:07:24 - (conf-yaml-loader.c:265) <Info> (ConfYamlParse
[5053] 9/2/2018 -- 11:07:24 - (suricata.c:1073) <Notice> (LogVersion) -- Thi
[5053] 9/2/2018 -- 11:08:05 - (tm-threads.c:2172) <Notice> (TmThreadWaitOnTh
[5053] 9/2/2018 -- 11:35:21 - (suricata.c:2716) <Notice> (SuricataMainLoop)
[5076] 9/2/2018 -- 11:35:30 - (source-pcap-file.c:354) <Notice> (ReceivePcap
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
99.90  749.382002        3641    205839           nanosleep
 0.10    0.731366           5    143718       113 futex
 0.00    0.006696          14       495           munmap
 0.00    0.006093           1      5423           brk
 0.00    0.002849         407         7           madvise
 0.00    0.000695           0      4865         2 read
 0.00    0.000531           1       589           mmap
 0.00    0.000288           3       100           mprotect
 0.00    0.000208           7        32           clone
 0.00    0.000173           2        74           write
 0.00    0.000161           2        69           open
 0.00    0.000101           3        34        34 access
 0.00    0.000093           2        60           fstat
 0.00    0.000090           1        69           close
 0.00    0.000021           1        22         2 stat
```

```
 0.00    0.000008           4         2           unlink
 0.00    0.000008           0        74           gettid
 0.00    0.000005           1        10           rt_sigaction
 0.00    0.000005           2         3           rt_sigprocmask
 0.00    0.000005           5         1           bind
 0.00    0.000003           3         1           socket
 0.00    0.000003           1         3           prctl
 0.00    0.000003           3         1           arch_prctl
 0.00    0.000002           1         3           lseek
 0.00    0.000002           1         2         2 ioctl
 0.00    0.000002           1         2           getrlimit
 0.00    0.000002           1         2           sysinfo
 0.00    0.000002           2         1           set_tid_address
 0.00    0.000001           1         1           listen
 0.00    0.000001           1         1           setsockopt
 0.00    0.000001           1         1           chmod
 0.00    0.000001           1         1           setrlimit
 0.00    0.000001           1         1           set_robust_list
 0.00    0.000000           0         1           execve
 0.00    0.000000           0         1           uname
 0.00    0.000000           0         6           getdents
 0.00    0.000000           0        14        14 mkdir
------ ----------- ----------- --------- --------- ----------------
100.00  750.131422                361528       167 total
```

pcap run 2

Using:

- 160GB pcap live ISP capture
- 30k ETPro rules

NOTE: This is the same run as above. The purpose of this second identical run is to show us if there are some small deviations and where to expect those so that we can have that in mind when we do the runs over the patched kernel.

Command:

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k none \
-r /var/log/suricata/pcap/bigtest.pcap --runmode=autofp \
&> strace-bigpcap-run-2
```

Output:

```
[6948] 9/2/2018 -- 11:35:32 - (conf-yaml-loader.c:265) <Info> (ConfYamlParse
[6948] 9/2/2018 -- 11:35:32 - (suricata.c:1073) <Notice> (LogVersion) -- Thi
[6948] 9/2/2018 -- 11:36:22 - (tm-threads.c:2172) <Notice> (TmThreadWaitOnTh
```

```
[6948] 9/2/2018 -- 12:03:36 - (suricata.c:2716) <Notice> (SuricataMainLoop)
[7005] 9/2/2018 -- 12:03:45 - (source-pcap-file.c:354) <Notice> (ReceivePcap
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
99.91  751.687483        3661    205336           nanosleep
 0.09    0.680834           5    143158        87 futex
 0.00    0.009765          20       498           munmap
 0.00    0.006480           1      5423           brk
 0.00    0.001268           2       589           mmap
 0.00    0.000921           0      4865         2 read
 0.00    0.000741          93         8           madvise
 0.00    0.000501           7        69           open
 0.00    0.000444           4       100           mprotect
 0.00    0.000217           3        74           write
 0.00    0.000217           7        32           clone
 0.00    0.000163           5        34        34 access
 0.00    0.000162           3        60           fstat
 0.00    0.000139           2        69           close
 0.00    0.000043           2        22         2 stat
 0.00    0.000017           6         3           prctl
 0.00    0.000013           1        10           rt_sigaction
 0.00    0.000010           5         2           getrlimit
 0.00    0.000008           4         2           unlink
 0.00    0.000007           2         3           rt_sigprocmask
 0.00    0.000006           3         2         2 ioctl
 0.00    0.000006           6         1           bind
 0.00    0.000006           6         1           setrlimit
 0.00    0.000006           0        74           gettid
 0.00    0.000004           1         3           lseek
 0.00    0.000003           2         2           sysinfo
 0.00    0.000003           3         1           arch_prctl
 0.00    0.000003           3         1           set_tid_address
 0.00    0.000003           3         1           set_robust_list
 0.00    0.000002           2         1           socket
 0.00    0.000002           2         1           chmod
 0.00    0.000001           1         1           listen
 0.00    0.000001           1         1           setsockopt
 0.00    0.000001           1         1           uname
 0.00    0.000000           0         1           execve
 0.00    0.000000           0         6           getdents
 0.00    0.000000           0        14        14 mkdir
------ ----------- ----------- --------- --------- ----------------
100.00  752.389480              360469       141 total
```

pcap run 3

Using:

- 8.1GB pcap live ISP capture
- 30k ETPro rules

NOTE: This is the same run as above but on different pcap size. The purpose of this second identical run is to show us if there are some small deviations and where to expect those so that we can have that in mind when we do the runs over the patched kernel.

Command:

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k none \
-r /var/log/suricata/pcap/test.pcap --runmode=autofp \
&> strace-smallpcap-run-1
```

Output:

```
[4680] 9/2/2018 -- 11:03:33 - (conf-yaml-loader.c:265) <Info> (ConfYamlParse
[4680] 9/2/2018 -- 11:03:33 - (suricata.c:1073) <Notice> (LogVersion) -- Thi
[4680] 9/2/2018 -- 11:04:24 - (tm-threads.c:2172) <Notice> (TmThreadWaitOnTh
[4680] 9/2/2018 -- 11:05:35 - (suricata.c:2716) <Notice> (SuricataMainLoop)
[4739] 9/2/2018 -- 11:05:43 - (source-pcap-file.c:354) <Notice> (ReceivePcap
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
94.55   35.217995         722     48761           nanosleep
 5.19    1.934846          24     81204        80 futex
 0.18    0.065717        2987        22         2 stat
 0.03    0.009975        4988         2           unlink
 0.02    0.008279           1      5932           brk
 0.02    0.005664          11       502           munmap
 0.01    0.002228         557         4           madvise
 0.00    0.001004           0      4865         2 read
 0.00    0.000830           1       589           mmap
 0.00    0.000546           7        74           write
 0.00    0.000415          13        32           clone
 0.00    0.000270           4        74           gettid
 0.00    0.000170           2       100           mprotect
 0.00    0.000130           2        69           open
 0.00    0.000074           7        10           rt_sigaction
 0.00    0.000070           1        60           fstat
 0.00    0.000033          33         1           socket
 0.00    0.000029          10         3           rt_sigprocmask
 0.00    0.000026           0        69           close
 0.00    0.000020           3         6           getdents
```

```
 0.00    0.000012           6         2         2 ioctl
 0.00    0.000008           3         3           lseek
 0.00    0.000006           6         1           bind
 0.00    0.000006           2         3           prctl
 0.00    0.000005           5         1           chmod
 0.00    0.000004           2         2           getrlimit
 0.00    0.000003           3         1           setsockopt
 0.00    0.000003           3         1           set_tid_address
 0.00    0.000003           3         1           set_robust_list
 0.00    0.000001           1         1           listen
 0.00    0.000000           0        34        34 access
 0.00    0.000000           0         1           execve
 0.00    0.000000           0         1           uname
 0.00    0.000000           0        14        14 mkdir
 0.00    0.000000           0         2           sysinfo
 0.00    0.000000           0         1           arch_prctl
 0.00    0.000000           0         1           setrlimit
------ ----------- ----------- --------- --------- ----------------
100.00   37.248372                142449       134 total
```

pcap run 4

Using:

- 8.1GB pcap live ISP capture
- 30k ETPro rules

NOTE: This is the same run as above. The purpose of this second identical run is to show us if there are some small deviations and where to expect those so that we can have that in mind when we do the runs over the patched kernel.

Command:

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k none \
-r /var/log/suricata/pcap/test.pcap --runmode=autofp \
&> strace-smallpcap-run-2
```

Output:

```
[4855] 9/2/2018 -- 11:05:45 - (conf-yaml-loader.c:265) <Info> (ConfYamlParse
[4855] 9/2/2018 -- 11:05:45 - (suricata.c:1073) <Notice> (LogVersion) -- Thi
[4855] 9/2/2018 -- 11:06:39 - (tm-threads.c:2172) <Notice> (TmThreadWaitOnTh
[4855] 9/2/2018 -- 11:07:17 - (suricata.c:2716) <Notice> (SuricataMainLoop)
[4918] 9/2/2018 -- 11:07:23 - (source-pcap-file.c:354) <Notice> (ReceivePcap
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
98.33   26.938375         667     40380           nanosleep
```

```
 1.59    0.435495           5     81020       170 futex
 0.04    0.010092           2      5923           brk
 0.02    0.005007          10       507           munmap
 0.02    0.004506         644         7           madvise
 0.00    0.000815           1       589           mmap
 0.00    0.000676           0      4865         2 read
 0.00    0.000287           9        32           clone
 0.00    0.000284           3       100           mprotect
 0.00    0.000189           3        74           write
 0.00    0.000083           1        69           open
 0.00    0.000068           1        74           gettid
 0.00    0.000058           1        69           close
 0.00    0.000055           1        60           fstat
 0.00    0.000044           1        34        34 access
 0.00    0.000025           3        10           rt_sigaction
 0.00    0.000018           6         3           rt_sigprocmask
 0.00    0.000016           1        22         2 stat
 0.00    0.000012           2         6           getdents
 0.00    0.000009           9         1           listen
 0.00    0.000007           2         3           lseek
 0.00    0.000006           3         2         2 ioctl
 0.00    0.000005           3         2           unlink
 0.00    0.000005           2         3           prctl
 0.00    0.000004           2         2           getrlimit
 0.00    0.000004           4         1           arch_prctl
 0.00    0.000003           3         1           socket
 0.00    0.000003           3         1           bind
 0.00    0.000003           3         1           set_tid_address
 0.00    0.000003           3         1           set_robust_list
 0.00    0.000001           1         1           chmod
 0.00    0.000000           0         1           setsockopt
 0.00    0.000000           0         1           execve
 0.00    0.000000           0         1           uname
 0.00    0.000000           0        14        14 mkdir
 0.00    0.000000           0         2           sysinfo
 0.00    0.000000           0         1           setrlimit
------ ----------- ----------- --------- --------- ----------------
100.00   27.396158                133883       224 total
```

Patched kernel - running live

Using: - 30k ETPro rules

We measured Suricata running live process for 2 hours 3 different sample times during a 24 hr period this time on patched kernel.

Command:

```
perf stat -e 'syscalls:sys_enter_*' \
-p 6732 -a sleep 7200 &> suri.syscalls-1
```

Output:

```
root@suricata:/home/pevman/tests/meltdown/postpatched-kernel#
cat suri.syscalls-1 |grep -v ' 0 '|grep syscalls | sort -rn
       84,924,210      syscalls:sys_enter_bpf
       68,532,207      syscalls:sys_enter_write
       49,748,847      syscalls:sys_enter_poll
       31,703,441      syscalls:sys_enter_futex
        1,405,306      syscalls:sys_enter_nanosleep
          114,988      syscalls:sys_enter_getsockopt
           90,183      syscalls:sys_enter_mprotect
           35,945      syscalls:sys_enter_select
           11,136      syscalls:sys_enter_getdents
            5,571      syscalls:sys_enter_open
            5,571      syscalls:sys_enter_close
            5,570      syscalls:sys_enter_newfstat
               12      syscalls:sys_enter_mkdir
                5      syscalls:sys_enter_mmap
                3      syscalls:sys_enter_munmap
                3      syscalls:sys_enter_getrandom
                2      syscalls:sys_enter_madvise
                2      syscalls:sys_enter_lseek
                1      syscalls:sys_enter_read

root@suricata:/home/pevman/tests/meltdown/postpatched-kernel#
cat suri.syscalls-2 |grep -v ' 0 '|grep syscalls | sort -rn
       80,899,986      syscalls:sys_enter_bpf
       56,413,573      syscalls:sys_enter_write
       52,876,878      syscalls:sys_enter_poll
       26,021,319      syscalls:sys_enter_futex
        1,403,290      syscalls:sys_enter_nanosleep
          114,841      syscalls:sys_enter_getsockopt
           35,941      syscalls:sys_enter_select
           11,120      syscalls:sys_enter_getdents
            5,562      syscalls:sys_enter_open
            5,562      syscalls:sys_enter_newfstat
            5,562      syscalls:sys_enter_close
              887      syscalls:sys_enter_mprotect
               12      syscalls:sys_enter_mkdir
                2      syscalls:sys_enter_lseek

root@suricata:/home/pevman/tests/meltdown/postpatched-kernel#
cat suri.syscalls-3 |grep -v ' 0 '|grep syscalls | sort -rn
       75,524,061      syscalls:sys_enter_bpf
```

```
50,218,601      syscalls:sys_enter_write
48,793,033      syscalls:sys_enter_poll
26,645,822      syscalls:sys_enter_futex
 1,403,810      syscalls:sys_enter_nanosleep
   113,632      syscalls:sys_enter_getsockopt
    35,942      syscalls:sys_enter_select
    11,130      syscalls:sys_enter_getdents
     5,567      syscalls:sys_enter_open
     5,567      syscalls:sys_enter_newfstat
     5,567      syscalls:sys_enter_close
       131      syscalls:sys_enter_mprotect
        12      syscalls:sys_enter_mkdir
         3      syscalls:sys_enter_munmap
         2      syscalls:sys_enter_mmap
         2      syscalls:sys_enter_lseek
         2      syscalls:sys_enter_getrandom
```

Observations

In the results above with regards to `strace` we have in the first case running Suricata in regular `af-packet mode` with `cluster-type: cluster_flow` and the second run (with the patched kernel) we have Suricata running with XDP and `af-packet cluster-type: cluster_qm`. It was evident with the usage of `syscalls:sys_enter_bpf`.

Overall there is no definitive observation of any performance penalty (especially in the range of additional 5-25% CPU usage) with our set up from the tests done. There is also no observation of any adverse impact of general performance of Suricata with regards to packet drops/memcap hits or higher CPU usage.

Those observations have been confirmed in production at Mozilla. We have been running Meltdown patched kernel from the day of release, and we saw no noticeable performance degradation.

Technical details

The Meltdown mitigation (it does not solve the problem, it makes it impossible to abuse it) is KPTI (for Linux) and KvaShadow (for Windows). What this mitigation does is it removes most of the kernel mappings from the userspace (userspace mappings in the kernel page tables are still there).

Now that most of those mappings are gone, every system call needs a full or partial page table reload, flushing some or all data from the TLB (and some or all data from the instruction TLB, which is separate).

For amortizing the cost impact, it would be possible to use PCID (process context identifier) for CPUs that have it. Thought initially as a performance optimization for the VMEXIT case, when hypervisor was switching between virtual machines (thus

trashing the TLB frequently) can now be reused for the userspace -> kernel space switches.

While PCID is present on Intel platforms from Sandy Bridge up, only from Haswell up, they have INVPCID that can precisely remove individual entries from the TLB, instead of flushing the whole thing.

With PCID+INVPCID the performance impact of KPTI should not be noticeable. You want to run Haswell if you can.

As far as other mitigations go - IBRS + IBPB might have a noticeable performance impact on everything pre-Skylake. We didn't test it because no stable CPU microcode was available.

Given all these and what we know about those attacks, one might wonder...

Should I be worried?

In which we walk our readers through a beautiful world of risk management.

Microsoft released a wonderful example of risk management related to those vulnerabilities

We highly recommend to read it and take a while to think about scenarios here.

The NSM sensor, by design, should be on an isolated host that does not expose any interfaces but the bare minimum. Logs should be shipped out, the only service should be SSH (from something like a bastion host), with MFA. No unnecessary services should be running.

Should our reader be worried about highly scientific and unlikely attacks that require for the threat actor to be on the same host? Are there other ways the threat actor can accomplish what they want if they already have a local shell?

The Linux local privilege escalation history looks exploitable nonetheless and as such it is safe to assume there are many, much more comfortable, ways to escalate local privileges than using Meltdown and Spectre attacks on a local host.

But you recommend using eBPF which was used in a Spectre PoC

Yes, we do. A careful reader will see that using eBPF was merely optimization the researches did and they had several other methods to find gadgets to abuse in the Spectre PoC. They just brought gadgets with them, in the form of a short eBPF program, to prove the point.

The eBPF code has been significantly hardened. The method researchers used to bypass verification (by calling a function that didn't verify the provided eBPF code) has been removed. JIT constant blinding is all over the place.

BTW, KVM broke the PoC in 5 seconds by clearing registers on VMEXIT :)

While there are for sure other vulnerabilities in the eBPF JIT, we would recommend doing a risk assessment of how likely it those would be used.

Chapter 4

Further reading

XDP:

[1] http://people.netfilter.org/hawk/presentations/LLC2017/XDP_DDoS_protecting_LLC2017.pdf

[2] http://people.netfilter.org/hawk/presentations/driving-IT2017/driving-IT-2017_XDP_eBPF_technology_Jesper_Brouer.pdf

[3] https://prototype-kernel.readthedocs.io/en/latest/blogposts/xdp25_eval_generic_xdp_tx.html

[4] http://people.netfilter.org/hawk/presentations/NetConf2017_Seoul/XDP_devel_update_NetConf2017_Seoul.pdf

[5] https://prototype-kernel.readthedocs.io/en/latest/networking/XDP/index.html

[6] https://www.iovisor.org/technology/xdp

[7] https://people.netfilter.org/hawk/presentations/NetDev2.2_2017/XDP_for_the_Rest_of_Us_Part_2.pdf

Netronome white papers

[8] https://open-nfp.org/dataplanes-ebpf/technical-papers/

symRSS:

[9] http://www.ndsl.kaist.edu/~kyoungsoo/papers/TR-symRSS.pdf

[10] http://www.ran-lifshitz.com/2014/08/28/symmetric-rss-receive-side-scaling/

[11] https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/intel-ethernet-flow-director.pdf

Suricata Readthedocs - XDP:

[12] http://suricata.readthedocs.io/en/latest/capture-hardware/ebpf-xdp.html?highlight=XDP#ebpf-and-xdp

Linux Syscalling:

[13] http://www.brendangregg.com/perf.html

Chapter 5

Authors

`In mob we trust`
A word about the authors:

Michal Purzynski (@MichalPurzynski)

- Threat Management, Mozilla
- Intrusion detection
- Digital Forensics
- Incident response

Peter Manev (@pevma)

- Suricata Core Team
- Lead QA and training instructor
- Stamus Networks
- Mobster evangelist

Chapter 6

Thank you

People and organizations without whom this guide would have not been possible:

- Eric Leblond (@regiteric – Suricata AFP/XDP godlike dev doing kernel patches while chasing off cats from the keyboard)
- Jesper Brouer (@netoptimiser, RedHat Principal kernel engineer, XDP developer)
- Dave Miller for AFPacket :-)
- IOvisor project
- SuriCon 2017 !!
- Suricata community for inspiring us to push the limits further