



SEPTun Mark II



# 第一章

## 介绍

SEPTun 是（希望至少）一年一度的系列文章，指导在高速网络中部署 Suricata IDS/IPS/NSM 的极端和实验技术。

在本文中，我们反思并分享了我们所学到的经验和知识，并详细说明了如何部署这些发现。除此之外，我们还描述了我们在这段旅程中遇到的挑战。

每个标志都总结了去年的标志，不包括与去年相比仍然有效的内容。如果有什么需要更新，我们在这里显示，如果没有 - 旧版本的文件仍然有效。如果您在阅读此内容之前还没有阅读 SEPTun Mark I，请先阅读。

准备？系好安全带，享受旅程吧！！当我们结束研究时 - 大 Meltdown 发生了。后续内核补丁的主要担忧之一是它会影响 15%+ 的性能。我们做了一些测试来确认这是否会对我们的设置产生影响，并在本文中记录了这些测试。

SEPTun Mark II 研究结果最初在 SuriCon 2017 上发表。

### 本指南与 Mark I 有何不同？

- eXpress 数据路径 - 描述它是什么以及如何使用它
- 我们收到了很多关于 RSS 的问题。RSS 部署指南在 - 在 Mark II 中出演

### 我们的设置

#### 内核

本指南已经过测试并确认运行良好（我们需要具有当前工具链和内核的东西）：

- Debian 测试（内核为 4.14.x/4.15.2 的 Buster）









要做到这一点，XDP 依赖于 eBPF 脚本。这些脚本是使用 JIT 编译的，并且速度非常快。注意 - 在最近的 Spectre 攻击中，您听到了很多关于 eBPF 的负面新闻。我们将在本文中解决这些问题，请继续阅读。

您可能想知道 eBPF 是什么，它与 Suricata 的 XDP 旁路有什么关系。

## eBPF

- 优于传统 BPF
- 允许每个数据包运行用户钩子/程序
- 扩展和改进 Suricata 的性能
- 打开更多内核空间可能性
- 大象流旁路
- 你需要编写自己的 eBPF “钩子”

在 linux 手册页中可以找到对 eBPF 的更好描述之一：

在 Linux 中，通常认为 eBPF 是 cBPF 的继承者，内核在内部将 cBPF 表达式转换为 eBPF 表达式并执行后者。它们的执行可以在解释器中执行，也可以在设置时执行，它们可以被即时编译（JIT）以作为本机机器代码运行。目前，x86\_64、ARM64、s390、ppc64 和 sparc64 架构支持 eBPF JIT，而 PPC、SPARC、ARM 和 MIPS 支持 cBPF，但尚未切换到 eBPF JIT 支持。

eBPF 的指令集具有与 cBPF 指令集相似的底层原理，但它的建模更接近底层架构，以更好地模拟原生指令集，以实现更好的运行时性能。它被设计为通过一对一的映射进行 JIT 化，这也为编译器提供了通过 eBPF 后端生成优化 eBPF 代码的可能性，该后端的执行速度几乎与原生编译的代码一样快。鉴于 LLVM 提供了这样的 eBPF 后端，因此可以很容易地用 C 语言的子集对 eBPF 程序进行编程。除此之外，eBPF 基础设施还带有一个名为“映射”的结构。eBPFmap 是在多个 eBPF 程序之间共享的键/值存储，也可以在 eBPF 程序和用户空间应用程序之间共享。

最显着的好处是，对于 XDP 旁路，Linux 内核不必创建 SKB（Linux 套接字缓冲区结构）——因此节省了创建以后会被丢弃的东西的 CPU 周期。不做不必要的工作可以提高性能。早点决定，早点放弃。

有三种模式可用（suricata.yaml 配置中的 af-packet 部分）：



- XDP 模式: 软
- XDP 模式: 司机
- XDP 模式: 硬件

“hw” 模式意味着数据包永远不会被 Linux 内核看到, 并且会在卡本身的硬件级别被丢弃。Intel 卡目前还不能做到这一点 (本文的作者)。Netronome 卡可以做到。

“驱动程序” 模式意味着数据包将在创建 SKB 之前在驱动程序级别被丢弃, 但在硬件中不会。Linux 内核将看到谁的数据包, 但会很早地丢弃它们, 从而节省大部分处理时间。带有来自 Linux 内核的驱动程序的 Intel 卡支持它。这些驱动程序的上游 Intel 版本 (来自 SourceForge) 不支持 XDP, 而且它们不太可能支持 XDP。

“软” 模式意味着数据包将在创建 SKB 后被丢弃, 然后 Suricata 才能使用这些数据包。“软” 模式是最慢的模式, 但也是一个很好的回退模式, 因为它不需要任何硬件或驱动程序支持。

在本文中, 我们使用 `xdp-mode:` 驱动程序。

## 支持本机驱动程序 XDP 的 NIC

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

`xdp-mode` 的 `hw` 目前仅在 Netronome 中可用  
文章写作。

## 先决条件

请注意, XDP 卸载需要对称 RSS 才能正常工作。我们使用 “QM” AF\_Packet 模式, 该模式将 Suricata 线程绑定到驱动程序队列, 并且 card 负责在这些队列之间对称地散列流。

如何配置对称 RSS 的说明将在本文后面进一步说明。有关如何配置系统固件 (BIOS/UEFI 设置很重要) 和 Linux 数据处理如何工作以及如何测量数据包丢失的一般信息, 请参阅 SEPTun Mark I。这里唯一变化的是对称 RSS。我们将推荐相同的内核和驱动程序设置, 并将 Suricata worker 固定到核心, 并将所有可以从 worker 核心中移动的工作负载移动。





## 包

特定于我们的 Suricata 功能设置：

```
sudo apt-get -y install git build-essential autoconf automake \libtool pkg-  
configlibpcre3 libpcre3-dbg libpcre3-dev \libpcap-dev libnet1-devlibyaml-0-2  
libyaml-dev zlib1g \zlib1g-devlibmagic-dev libcap-ng-dev libjansson-dev  
\libjansson4 libnss3-dev libnspr4-dev libgeoip-dev liblua5.1-dev \rustc cargo
```



## 叮当声和精灵

确保您已在系统上为 XDP 安装了 clang & elf。默认的 clangversion 应该可以。

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

## 内核和 NIC 相关

- 支持 XDP 的较新内核（在本例中为 4.13.10+）
- 取决于 af 数据包模式（本文后面介绍） - RSS 对称  
在 NIC 上进行哈希处理（在本例中为 Intel 82599ES 10-Gigabit/x520/x540）
- 在树内核驱动程序中，NIC 驱动程序。（又名下载和编译您的驱动程序  
似乎不起作用 - 在文章后面解释了如何作）

如果您需要帮助，您可以在此处的脚本的帮助下构建所需的内核版本

## 禁用 irqbalance

```
systemctl stop irqbalancesystemctl disable irqbalance
```

（确保它已消失 :)从系统中）

## BPF

还需要修补的 BPF（标头）：

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



## 编译安装 Suricata

下面的 Suricata 配置/编译包括我们需要构建的一些额外功能（如 Rust/文件提取/geoip 等）。启用XDP功能的一个重要是配置行中的--enable-ebpf--build。  
下面我们使用 Suricata git master，但 XDP 功能存在于 Suricata 中

4.1+

```
git 克隆      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 ./配置 \
--prefix=/usr/ --sysconfdir=/etc/ --localstatedir=/var/ \
--with-libnss-libraries=/usr/lib \
--with-libnss-includes=/usr/include/nss/ \
--with-libspr-libraries=/usr/lib \
--with-libspr-includes=/usr/include/nspr \
--enable-geoip --enable-luajit --enable-rust \
--enable-ebpf --enable-ebpf-build
```

```
make clean & makesudomake
install-fullsudo ldconfig
```

根据需要复制生成的 xdp 过滤器 — 您可以在 suricata.yaml.In 中指定特定路径，在本例中为 `wwas/etc/suricata/`:

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

在 `suricata.yaml` 中设置 `af-packet` 部分/接口。我们将使用可以在 NIC 上进行对称哈希 `cluster_qmas xdp-mode: driver`，我们还将使用 `/etc/suricata/xdp_filter.bpf`（在我们的示例 TCP 卸载/旁路中）

在 `suricata.yaml` 配置中使用一个接口 -

```
- interface: eth3threads: 14cluster-id: 97cluster-
type: cluster_qm # 对称哈希
```

是必须的！

碎片整理: yes# 包含将插入内核的 'loadbalancer' 函数的 eBPF 文件# 并用作负载均衡函数  
#ebpf-lb-file: /etc/suricata/lb.bpf# 包含将入的 'filter' 函数的 eBPF 文件



```
# 内核并用作数据包过滤功能# 包含“xdp”函数的 eBPF 文件将插入到#内核中并用作 XDP 数
数据包过滤功能#ebpf-filter-file: /etc/suricata/filter.bpf# Xdp 模式, “soft”用于基于
skb 的版本, “driver”用于基于网卡# 的“driver”, “hw”用于支持 eBPF 的卡.xdp-
mode: driverxdp-filter-file: /etc/suricata/xdp_filter.bpf# 如果 ebpf 过滤器实现了旁
路功能, 你可以将# 'bypass' 设置为 yes 并从这些功能中受益bypass: yesuse-mmap:
yesmmap-locked: yes# 使用 tpacket_v3, 捕获模式, 仅当 user-mmap 为 truetpacket-v3
才激活tpacket-v3: yesring-size: 200000block-size: 1048576
```



#### 在 suricata.yaml 配置中使用两个接口 -

```
- 接口: eth3threads: 7cluster-id: 97cluster-type: cluster_qmdefrag: 是# 包含“负
载均衡器”函数的 eBPF 文件, 该函数将入# 内核并用作负载均衡函数#ebpf-lb-
file: /etc/suricata/lb.bpf# 包含“过滤器”函数的 eBPF 文件将入到#内核中并用作 XDP 数据
包过滤函数# 包含“xdp”函数的 eBPF 文件将插入到#内核中并用作 XDP 数据包过滤函数#ebpf-
filter-file: /etc/suricata/filter.bpf# Xdp模式, “soft”用于基于skb的版本, “driver”
用于基于网卡#的版本, “hw”用于支持eBPF的卡.xdp-mode: driverxdp-filter-
file: /etc/suricata/xdp_filter.bpf# 如果ebpf过滤器实现了旁路功能, 你可以设置# 'bypass'
为yes, 并从这些功能中受益bypass: yesuse-mmap: yesmmap-locked: yes# 使用tpacket_v3,
捕获模式, 仅当user-mmap为true时才激活tpacket-v3: yesring-size: 200000block-size:
1048576
```

- 接口: eth2线程: 7集群ID: 98集群类型: cluster\_qmdefrag: 是# 包含“负载均衡器”函数的 eBPF 文件, 该函数将入到内核中并用作负载均衡函数#ebpf-lb-  
file: /etc/suricata/lb.bpf# 包含“过滤器”函数的 eBPF 文件将入到#内核中并用作数据包过滤器# 包含“xdp”函数的 eBPF 文件#内核中并用作 XDP 数据包过滤函数#ebpf-filter-  
file: /etc/suricata/filter.bpf# Xdp模式, “soft”用于基于skb的版本, “driver”用于基于网卡#的版本, “hw”用于支持eBPF的卡.xdp-mode: driverxdp-filter-  
file: /etc/suricata/xdp\_filter.bpf# 如果ebpf过滤器实现了旁路功能, 你可以设置# 'bypass' 为yes, 并从这些功能中受益bypass: yesuse-mmap: yesmmap-locked: yes# 使用tpacket\_v3, 捕获模式, 仅当user-mmap为true时才激活tpacket-v3: yesring-size: 200000block-size: 1048576

此外, 在“stream”部分中启用

“bypass”:

流:

bypass: 真

设置的其中一个测试计算机的示例还包括 (stream.reassembly.deprh 的示例):

流:

内存cap: 14gbchecksum-  
validation: nobypass:  
yesprealloc-sessions: 375000  
内联: autoreassembly:

内存容量: 20GB深度:  
lmbtoservert-chunk-size:  
2560toclient-chunk-size:  
2560randomize-chunk-size: 是



随机块范围: 10row: 是分段预  
alloc: 200000

以及一些超时, 例如:

flow-timeouts 中:

违约:

新: 3已建立: 30已关闭: 0已绕  
过: 20紧急新: 1紧急已建立: 10  
紧急已关闭: 0紧急已绕过:  
5tcp:

新: 3已建立: 30已关闭: 1已绕  
过: 25紧急新: 1紧急已建立: 10  
紧急已关闭: 0紧急绕过: 5UDP :

新增: 3已建立: 30已绕过: 20紧  
急-新: 1紧急已建立: 10紧急已绕  
过: 5ICMP;

新增: 2已建立: 30已绕过: 20  
紧急-新: 1紧急已建立: 10紧  
急-绕过: 5





## 在 NIC 上设置对称哈希

### 原因

RSS 是最初由 Microsoft 定义的技术。它通过使用 IP 元组计算哈希值，在多个内核或 CPU 上对网络数据进行常规负载均衡。

从 IDS/IPS 的角度来看，问题在于它需要按照最终客户端的意愿来查看流量，才能正确地完成其工作。RSS 的挑战在于，它 RSS 是出于其他目的而设计的，例如扩展大型 Web/文件共享安装，因此不需要与 IDS/IPS 部署相同的“流”一致性。

正如 Suricata 文档中明确解释的那样：

“接收方缩放是网卡使用的一种技术，用于在 NIC 上的各种队列上分配传入流量。这是为了提高性能，但重要的是要认识到它是为平均流量设计的，而不是为 IDS 数据包捕获场景设计的。RSS 使用哈希算法在各种队列上分配传入流量。此哈希通常不对称。这意味着在接收流的两端时，每一端都可能位于不同的队列中。遗憾的是，在部署 Suricata 时，这是使用 span 端口或分流器时的典型情况。”

换句话说，在大多数 RSS 情况下 -

· ipsrc=1.1.1.1, ipdst=2.2.2.2, sport=11111, dport=22222

与

· ipsrc=2.2.2.2, ipdst=1.1.1.1, sport=22222, dport=11111

.... 所以不要去同一个队列/线程！事实证明，在特定的 Intel NIC 上，您可以使用低熵密钥启用对称 RSS。在这种情况下 - 我们可以将 AF\_PACKET 与 cluster\_qm 一起使用 - 所有通过网卡链接到 RSS 队列的数据包都发送到同一个套接字。这至少需要 Linux 3.14。（仅供参考 - 其他方法也可用）

请严格按照以下说明作，以获得所需的结果（每个接口 - 在本例中为 eth3）：

ifconfig eth3 已关闭

使用和加载 NIC 的树内核驱动程序注意：在本例中，使用的内核和源是 4.15.2

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



### 在 NIC 上启用对称哈希并设置队列

ifconfig eth3 down netool -L eth3 组合 14 ethtool -K eth3 rxhash onetool -K eth3 ntuple onifconfi  
up/set\_irq\_affinity 2-15 ethtool -X eth3 hkey 6D: 5A: 6D: 5A: 6D: 5A: 6D: 5A: 6D: 5A:  
5A: 6D: 5A: 6D: 5A: 6D: 5A: 6D: 5A: 6D: 5A: 6D: 5A: 6D: 5A: 6D: 5A: 6D: 5A: 5A等

Etul -x which3 on etul - 在哪个

在上述设置中，您可以自由使用任何最新的 `set_irq_affinity` 脚本。它可在任何 Intel x520/710 NIC 源驱动程序下载中获得。

请记住，组合的数量必须与以后使用的 `forset_irq_affinity` 的数量相匹配，因此如果您有 16 个队列，请将 `worker` 固定到 16 个线程。

我们建议为每个 NUMA 节点保留 1-2 个内核，用于不进行数据包处理的 OS 和 Suricata thread。

注意：我们使用特定的低熵密钥进行对称哈希。有关对称哈希设置研究的更多信息

## AMD公司

对于 AMD CPU 系统，建议（根据我们的测试）遵循上述一般设置，除了将 Suricata 工作线程专用/固定在与 NIC 不同的 NUMA 节点 CPU (lscpu 将向您展示哪些是这些) 上（这与我们对于基于 Intel CPU 的系统所做的相反）。这将利用 AMD Hypertransport 技术获得更好的性能。

例如：

建筑:	x86_64
CPU作模式:	32 位、64 位
字节顺序:	小端
CPU:	48
在线 CPU 列表:	0-47
每个内核的线程数:	2
每个插槽的内核数:	24
插座:	1
NUMA 节点:	4
供应商 ID:	正牌 AMD
CPU 系列:	23

```

型: 1
型号名称: AMD EPYC 7401 24 核处理器
步 进: 2
CPU 兆赫: 2000.000
CPU 最大 MHz: 2000.0000
CPU 最小 MHz: 1200.0000BogoMIPS: 3991.97
虚拟化: AMD-VL1d 缓存: 32KL1i 缓存:
64KL2 缓存: 512KL3 缓存: 8192K
NUMA 节点0
CPU: 0-5, 24-29
NUMA 节点 1 CPU: 6-11, 30-
35
NUMA 节点 2 CPU: 12-17, 36-41
NUMA 节点
3 CPU: 18-23, 42-47

```

因此，在上面的示例中，如果您的 NIC 位于 NUMA 0 上，您将执行以下作：

- 启用 6（或最多 12 个 RSS 的任何数字 - 您需要尝试一下看看什么是 RSS 最适合您的设置）
- 将这些中断固定到 NUMA 0 上的 CPU（与卡相同）
- 将 `cpu affinity` 与 `Suricata`（`suricata.yaml`）一起使用，并确保 `af-packet` 工作线程在除 NUMA 0 之外的任何设备上运行
- 使用 `cluster_flow`（`suricata.yaml` 中的 `af-packet` 配置部分）

## 禁用 NIC 卸载

```

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

```

## 尽可能平衡

尽量使用网络卡平衡（例如在脚本中 `youdo/add in`）：

```

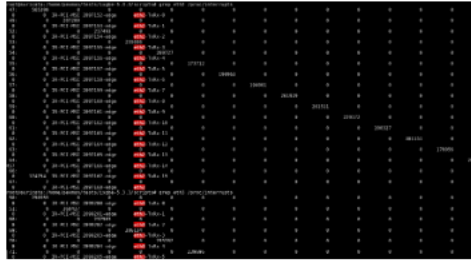
对于 tcp4 udp4 ah4 esp4 sctp4 tcp6 udp6 ah6 esp6 sctp6 中的原型； 做
    /sbin/ethtool -N eth3 rx-flow-hash $proto sdfn
做

```



## IRQ 亲和性

确保 irq 关联性正确。例



使用（示例）`grep eth2 /proc/interrupts` 检查

## 使用 XDP 启动 Suricata

确保您在 `suricata.yaml` 的第 `eve.json` 节中启用了统计信息：

- 统计：总计：是

线程数：否

deltas：是的

# 所有线程合并在一起的统计数据

# 每个线程的统计数据

# 包括 delta 值

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

确认您已在输出中使用 XDP 过滤器（示例）

```
(runmode-af-packet.c: 220) <Config> (解析AFPCConfig) \  
-- 在 iface eth3 上为 mmap 启用锁定内存 (runmode-af-packet.c: 231) <Config> (ParseAFPCConfig) \  
-- 在 iface eth3 上启用 tpacket v3 捕获 (runmode-af-packet.c: 326) <Config> (ParseAFPCConfig) \  
-- 对 AF_PACKET 使用基于队列的集群模式 (iface eth3) (runmode-af-packet.c: 424) <Info> (ParseAFPCConfig) \  
-- af-packet 将使用 '/etc/suricata/xdp_filter.bpf' 作为 XDP 过滤器文件 (runmode-af-packet.c: 429) <Config> (ParseAFPCConfig) \  
-- 对 AF_PACKET 使用旁路内核功能 (iface eth3) (runmode-af-packet.c: 609) <Config> (ParseAFPCConfig) \  
-- eth3: 使用数据发布调用 (util-runmodes.c: 296) <Info> (RunModeSetLiveCaptureWorkersForDevice) \  
-- 打算使用 8 个线程
```



## IRQ 亲和性

确保 irq 关联性正确。例

使用（示例）`grep eth2 /proc/interrupts` 检查

## 使用 XDP 启动 Suricata

确保您在 `suricata.yaml` 的第 `eve.json` 节中启用了统计信息：

```

- 统计：总计：是
                                # 所有线程合并在一起的统计数据
                                # 每个线程的统计数据
                                # 包括 delta 值
线程数：否
deltas：是的

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

```

确认您已在输出中使用 XDP 过滤器（示例）

```

(runmode-af-packet.c: 220) <Config> (解析AFPCConfig) \
-- 在 iface eth3 上为 mmap 启用锁定内存 (runmode-af-packet.c: 231) <Config>
(ParseAFPCConfig) \
-- 在 iface eth3 上启用 tpacket v3 捕获 (runmode-af-packet.c: 326) <Config>
(ParseAFPCConfig) \
-- 对 AF_PACKET 使用基于队列的集群模式 (iface eth3) (runmode-af-packet.c: 424)
<Info> (ParseAFPCConfig) \
-- af-packet 将使用 '/etc/suricata/xdp_filter.bpf' 作为 XDP 过滤器文件 (runmode-af-
packet.c: 429) <Config> (ParseAFPCConfig) \
-- 对 AF_PACKET 使用旁路内核功能 (iface eth3) (runmode-af-packet.c: 609) <Config>
(ParseAFPCConfig) \
-- eth3：使用数据发布调用 (util-runmodes.c: 296) <Info>
(RunModeSetLiveCaptureWorkersForDevice) \ 启用零复制模式
-- 打算使用 8 个线程

```





查看统计数据，看看你做得怎么样：

```
sudo 尾巴 -F /var/log/meerkat/eve.json |grep stats \
|jq 'select (.event_type=="stats") \|\{绕过: .stats.flow_bypassed,
bytes: .stats.decoder.bytes, \bytes_delta: .stats.decoder.bytes_delta, 百
分比: \ (.stats.flow_bypassed.bytes / .stats.decoder.bytes * 100)\}' -
```

您可以/应该立即看到明显的差异（例如，在我们的一次测试运行中，大约10%）：

```
{
  "绕过": {
    "closed": 3608681,
    "closed_delta": 6734, "pkts":
537177957, "pkts_delta": 604972,
    "bytes": 575615096046,
    "bytes_delta": 526741771},
    "bytes": 5278384812258,
    "bytes_delta": 3717316268,
    "percent": 10.905137016711027}{
```

```
  "绕过": {
    "已关闭": 3614378,
    "closed_delta": 5697, "pkts":
537888772, "pkts_delta": 710815,
    "bytes": 576283121574,
    "bytes_delta": 668025528},
    "bytes": 5281159632142,
    "bytes_delta": 2774819884,
    "percent": 10.912056474616803}
```

当然，根据流量的类型 - 百分比可能会更高。

## 优

点. XDP 通过引入- 为原生 Linux 驱动程序提供了显著的性能提升  
将 XDP 旁路功能转移到 Suricata，这反过来又允许在关键数据包路径中  
更早地处理大象流。从而减轻了 Suricata 和内核在旁路功能方面的重要  
工作



查看统计数据，看看你做得怎么样：

```
sudo 尾巴 -F /var/log/meerkat/eve.json |grep stats \  
|jq 'select (.event_type=="stats") \{|绕过: .stats.flow_bypassed,  
bytes: .stats.decoder.bytes, \bytes_delta: .stats.decoder.bytes_delta, 百  
分比: \ (.stats.flow_bypassed.bytes / .stats.decoder.bytes * 100)\}' -
```

您可以/应该立即看到明显的差异（例如，在我们的一次测试运行中，大约 10%）：

```
{  
  "绕过": {  
    "closed": 3608681,  
    "closed_delta": 6734, "pkts":  
537177957, "pkts_delta": 604972,  
    "bytes": 575615096046,  
    "bytes_delta": 526741771},  
    "bytes": 5278384812258,  
    "bytes_delta": 3717316268,  
    "percent": 10.905137016711027} {
```

```
  "绕过": {  
    "已关闭": 3614378,  
    "closed_delta": 5697, "pkts":  
537888772, "pkts_delta": 710815,  
    "bytes": 576283121574,  
    "bytes_delta": 668025528},  
    "bytes": 5281159632142,  
    "bytes_delta": 2774819884,  
    "percent": 10.912056474616803}
```

当然，根据流量的类型 - 百分比可能会更高。

## 优点

- XDP 通过引入 - 为原生 Linux 驱动程序提供了显著的性能提升  
将 XDP 旁路功能转移到 Suricata，这反过来又允许在关键数据包路径中更早地处理大象流。从而减轻了 Suricata 和内核在旁路功能方面的重要工作



流，然后 Suricata 处理它们 - 最大限度地减少需要完成的性能密集型工作。

## 警告

- 当前的 Suricata XDP eBPF 实现允许仅 TCP 流旁路
- 可以完成其他 eBPF 过滤器，尽管它需要您编写过滤器

## 错误和信息

在测试和研究过程中，发现了一些错误/优化，并由 Eric Leblond (@regiteric) 和 PeterManev (@pevma) 提交的有关 Linux 内核的补丁。Jesper Brouer (@netoptimizer) 也引入了一些进一步的补丁（和测试），这些补丁也有帮助。

## 宾果游戏

hw NIC 中断。

截至撰写本文时，最严重的错误之一是 IRQsreset 错误（在某些内核版本/Intel NIC 组合上）。似乎在 Suricata 启动并注入 eBPF 脚本代码之后，所有中断都被固定到 CPU0（我们需要将它们传播 - 不是那样的）。

注意：在我们的测试中（使用本文开头指定的 NIC），它显示它影响了内核 4.13.10，也可能影响了 4.14.x/4.15RC。使用内核

4.15.x (4.15.2) 我们没有问题。在 Suricata 以 - grep eth2 /proc/interrupts 开头后，您可以快速检查中断是否正确分布（或固定到 CPU0）（请参阅上面的 IRQaffinity 部分）。建议您对每个内核/NIC 驱动程序升级进行检查。

## 当前宾果修复

解决方法（如果您不是 4.15.2 及更高版本）是在 Suricata 启动后立即重新映射中断（如上所述使用 set\_irq\_affinity 脚本）。截至目前，Intel 还没有设计出明确的解决方案。

## 使用 cpumap 的 XDP 旁路来救援（如果需要）

为了更永久地解决上述问题（以及没有对称 RSS 可用性的情况） - xdp\_cpumap 概念代码由 Eric Leblond 于 2018 年 1 月移植并引入 Sulicata。xdp\_cpumap 是最近的 Linuxkernel xdp 功能，它（除其他外）支持在不中断流程的情况下将 XDP 帧重定向到 remoteCPU（第 13 页）。



流，然后 Suricata 处理它们 - 最大限度地减少需要完成的性能密集型工作。

## 警告

- 当前的 Suricata XDP eBPF 实现允许仅 TCP 流旁路
- 可以完成其他 eBPF 过滤器，尽管它需要您编写过滤器

## 错误和信息

在测试和研究过程中，发现了一些错误/优化，并由 Eric Leblond (@regiteric) 和 PeterManev (@pevma) 提交的有关 Linux 内核的补丁。Jesper Brouer (@netoptimizer) 也引入了一些进一步的补丁（和测试），这些补丁也有帮助。

## 宾果游戏 bug

Intel NIC 中断。

截至撰写本文时，最严重的错误之一是 IRQsreset 错误（在某些内核版本/Intel NIC 组合上）。似乎在 Suricata 启动并注入 eBPF 脚本代码之后，所有中断都被固定到 CPU0（我们需要将它们传播 - 不是那样的。

注意：在我们的测试中（使用本文开头指定的 NIC），它显示它影响了内核 4.13.10，也可能影响了 4.14.x/4.15RC。使用内核

4.15.x (4.15.2) 我们没有问题。在 Suricata 以 - grep eth2 /proc/interrupts 开头后，您可以快速检查中断是否正确分布（或固定到 CPU0）（请参阅上面的 IRQaffinity 部分）。建议您对每个内核/NIC 驱动程序升级进行检查。

## 当前宾果修复

解决方法（如果您不是 4.15.2 及更高版本）是在 Suricata 启动后立即重新映射中断（如上所述使用 set\_irq\_affinity 脚本）。截至目前，Intel 还没有设计出明确的解决方案。

## 使用 cpumap 的 XDP 旁路来救援（如果需要）

为了更永久地解决上述问题（以及没有对称 RSS 可用性的情况） - xdp\_cpumap 概念代码由 Eric Leblond 于 2018 年 1 月移植并引入 Sulicata。xdp\_cpumap 是最近的 Linux kernel xdp 功能，它（除其他外）支持在不中断流程的情况下将 XDP 帧重定向到 remoteCPU（第 13 页）。





## 如何使用

如果没有可用的 RSS 对称哈希，您可以尝试 xdp\_cpumap 确保您使用的是 Linux 内核 4.15.+

要使用 xdp\_cpumap 功能，您需要：

- 使用 xdp-cpu-redirect: [xx-xx] (af\_packet suricata.yaml 的部分)
- 使用 cluster-type: cluster\_cpu (suricata.yaml 的 af\_packet 部分)
- 确保将 NIC 的所有中断都固定到一个 CPU 上 (确保它在与 NIC 上的 NUMA 相同)
- 在相应 NUMA 节点的内核上启用并使用 CPU 关联性对于 NIC 位置核心。

例：

# Suricata 是多线程的。这里的线程可以受到影响。

```
set-cpu-affinity: yes# 调整线程的 cpu 亲和性。每个线程系列都可以在特定的 CPU 上绑定# 这两个适用于所有运行模式: # management-cpu-set 用于流超时处理, 计数器# worker-cpu-set 用于 'worker' 线程## 此外, 对于 autofp, 这些 apply: # receive-cpu-set 用于捕获线程# verdict-cpu-set 用于 IPS 判定线程#cpu-affinity:
```

```
- management-cpu-set: cpu: [ "all" ]# 在 affinity settings 中仅包含这些
cpuprio: default: "low"
```

```
- 接收-cpu-set: cpu:
[ 2-11 ] # 在 affinity 设置中仅包含这些 CPU
- worker-cpu-set: #cpu: [ "2-11" ]mode: "exclusive"# 显式使用 3 个线程,
不使用# detect-thread-ratio variable: # threads: 3prio: default: "high"
```



## 如何使用

如果没有可用的 RSS 对称哈希，您可以尝试xdp\_cpumap确保您使用的是 Linux 内核 4.15.+

要使用 xdp\_cpumap 功能，您需要：

- 使用 xdp-cpu-redirect: [xx-xx] (af\_packet suricata.yaml 的部分)
- 使用 cluster-type: cluster\_cpu (suricata.yaml 的af\_packet部分)
- 确保将 NIC 的所有中断都固定到一个 CPU 上 (确保它在与 NIC 上的 NUMA 相同)
- 在相应 NUMA 节点的内核上启用并使用 CPU 关联性对于 NIC 位置核心。

例：

# Suricata 是多线程的。这里的线程可以受到影响。

```
set-cpu-affinity: yes# 调整线程的 cpu 亲和性。每个线程系列都可以在特定的 CPU 上  
绑定# 这两个适用于所有运行模式: # management-cpu-set 用于流超时处理, 计数器#  
worker-cpu-set 用于 'worker' 线程## 此外, 对于 autofp, 这些 apply: # receive-  
cpu-set 用于捕获线程# verdict-cpu-set 用于 IPS 判定线程#cpu-affinity:
```

```
- management-cpu-set: cpu: [ "all" ]# 在 affinity settings中仅包含这些  
cpuprio: default: "low"
```

```
- 接收-cpu-set: cpu:  
[ 2-11 ] # 在 affinity 设置中仅包含这些 CPU  
- worker-cpu-set: #cpu: [ "2-11" ]mode: "exclusive"# 显式使用 3 个线程,  
不使用# detect-thread-ratio variable: # threads: 3prio: default: "high"
```





```
#- verdict-cpu-set: #cpu:
[ 0 ]#prio: #default:
“高”
```

## 工具

研究过程中使用的一些漂亮的工具：

- 使用 nstat 快速了解内核网络统计信息（感谢 @netoptimizer）

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

- 使用 mpstat 来获取 CPU 使用率和每个 CPU 的 IRQ 负载的概览。这命令将每 2 秒打印一次摘要。

```
mpstat -P 全部 -u -I scpu -I 总和 2
```

```
# 修复白色背景终端中的 mpstat 颜色: export S_COLORS_SGR='H=31;1: I=35;22: M=34;1: N=34;1: Z=32; 22'
```

- smp\_affinity\_list

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

- 返回顶部

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

- 非常有用的 NIC 统计工具（再次感谢 @netoptimizer）

[https://github.com/netoptimizer/network-testing/blob/master/bin/ethtool\\_stats.pl](https://github.com/netoptimizer/network-testing/blob/master/bin/ethtool_stats.pl)

- pidstat（再次感谢 @netoptimizer）

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

- 跟踪

```
strace -c your_comand_here
```

- perf top/stats - 如果没有这些:)你会怎么做

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

```
性能顶部 -C 0...
```



## 第 3 章

# The Meltdown 和 Spectre

当我们完成测试和实验时，Suricata 和 XDP 旁路 - bigMeltdown 发生了。

安全漏洞已修补，但人们普遍预期补丁本身会损害任何 Linux 系统上的系统调用性能。性能影响预期在 5-25% 之间，具体取决于特定的应用程序使用情况。我们很好奇 Suricata 和最新的 XDP 代码添加可能对这些影响 - 因此我们进行了一些测量。

在 Suricata 的情况下，我们进行了两种类型的测量：

- 在实时流量上运行
- 读取 pcaps

Suricata 实时运行时测量的第一个挑战是其性能取决于实际流量和流量类型 - 即使您进行连续/重复测量，也会有一些预期的偏差。

第二个挑战是测量工具本身会引入性能开销，因此我们不能只关注性能 - 而是查看在系统调用方面是否存在任何明显和令人震惊的偏差。

我们做了一些基准测试，如下所述。注意：用于进行测试运行的 Suricata 是使用调试符号 en-abled 编译的。再加上 strace 的使用，单独就会对性能产生影响。因此，我们对它的完成速度不感兴趣，而是对修补的内核和未修补的内核之间在系统调用（持续时间/发生次数）方面是否存在更大的偏差感兴趣。

此处解释了 Strace 开销（部分 - perf 与 strace）修补前后内核中使用的 Suricata 版本：

```
root@suricata: /home/pevman/tests/kernel# suricata --build-info这是 Suricata
版本 4.1.0-dev (rev f815027) 功能: PCAP_SET_BUFF PF_RING AF_PACKET
HAVE_PACKET_FANOUTLIBCAP_NG LIBNET1.1 HAVE_HTTP_URI_NORMALIZE_HOOK PCRE_JIT
HAVE_NSSHAVE_LUA HAVE_LUAJIT HAVE_LIBJANSSON TLS MAGIC RUST
```





SIMD 支持: SSE\_4\_2 SSE\_4\_1 SSE\_3Atomic 个实例: 1 2 4 8 16 字节 64 位, Little-endian 架构GCC 版本 4.2.1 兼容 Clang 3.8.0 (tags/RELEASE\_380/final), C 版本 199901编译\_FORTIFY\_SOURCE=OL1 缓存行大小 (CLS)=64线程 本地存储方法: \_\_threadcompiled 使用 LibHTP v0.5.25, 与 LibHTP v0.5.25 链接

Suricata 配置:

AF_PACKET支持:	是的
eBPF 支持:	是的
XDP 支持:	是的
PF_RING支持:	是的
NFQueue 支持:	不
NFLOG 支持:	不
IPFW 支持:	不
Netmap 支持:	不
已启用 DAG:	不
已启用 Napatech:	不
已启用 Unix 套接字:	是的
已启用检测:	是的
Libmagic 支持:	是的
libnss 支持:	是的
libnspr 支持:	是的
libjansson 支持:	是的
liblzma 支持:	是的
Hiredis 支持:	不
hiredis 与 libevent 异步:	不
Prelude 支持:	不
PCRE jit:	是的
LUA 支持:	是的, 通过 LuaJIT
自由主义 者:	是的
libgeoip 中:	是的
非捆绑 http:	不
旧 barnyard2 支持:	不
Hyperscan 支持:	是的
Libnet 支持:	是的
Rust 支持 (实验性):	是的
实验性 Rust 解析器:	不
Rust 严格模式:	是的
Rust 调试模式:	不



Suricatasc install:	是的
已启用分析:	不
已启用分析锁:	不
开发设置:	
瓢虫/补丁:	是的
已启用单元测试:	不
已启用调试输出:	不
已启用调试验证:	不
通用构建参数:	
安装前缀:	/usr/local
配置目录:	/usr/local/etc/suricata/
日志目录:	/usr/local/var/log/meerkat/
--前缀	/usr/local
--sysconfdir	/usr/local/etc 中
--localstate	/usr/local/var 中
主机:	x86_64-pc-linux-gnu
编译器:	clang-3.8 (执行名称) / clang (REA
已启用 GCC 保护:	不
已启用 GCC march 本机:	是的
已启用 GCC 配置文件:	不
启用位置无关可执行文件: noCFLAGS-ggdb -O0 -march=native	
-I\${srcdir}/../rust/gen/c-headersPCAP_CFLAGS	-I/usr/include
SECCFLAGS	

## 未修补的内核 - 读取 pcaps

root@suricata: /home/pevman/tests/kernel# uname -a  
Linux suricata 4.13.10-amd64 #1 SMP 星期一 2017 年 10 月 30 日 23: 01: 00 x86\_64x86\_64 x86\_64  
GNU/Linux

### pcap 运行 1

用:

- 160GB pcap 实时 ISP 捕获
- 30k ETPro 规则



命令:

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k 无 \
-r /var/log/suricata/pcap/bigtest.pcap --runmode=autofp \&> strace-bigpcap-run-1
```

输出:

```
[12087] 2018 年 9 月 2 日 -- 00: 53: 10 - (conf-yaml-loader.c: 265) <Info>
(ConfYamlPars[12087] 2018 年 9 月 2 日 -- 00: 53: 10 - (suricata.c: 1073) <Notice>
(LogVersion) -- Th[12087] 2018 年 9 月 2 日 -- 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% timesecondsusecs/callcallerrors syscall
```

99.92	699.882916	3433	203893		纳米睡眠
0.07	0.519546	4	140138	158	福特斯
0.00	0.011924	23	512		蒙地图
0.00	0.006752	614	11		臀部
0.00	0.005745	1	5423		BRK
0.00	0.000887	2	589		mmap
0.00	0.000765	0	4865	2	阅读
0.00	0.000456	5	100		mprotect 公司
0.00	0.000320	5	69		打开
0.00	0.000225	7	34	34	访问
0.00	0.000194	3	69		关闭
0.00	0.000169	3	60		fstat
0.00	0.000150	5	32		克隆
0.00	0.000047	1	74		写
0.00	0.000005	5	1		arch_prctl
0.00	0.000003	3	1		捆
0.00	0.000003	2	2		取消链接
0.00	0.000001	0	22	2	属性
0.00	0.000000	03			LSEEK
0.00	0.000000	0	10		rt_sigaction
0.00	0.000000	03			rt_sigprocmask
0.00	0.000000	02		2	个 ioctl
0.00	0.000000	01			插座
0.00	0.000000	01			听
0.00	0.000000	01			setsockopt
0.00	0.000000	01			执行
0.00	0.000000	01			uname
0.00	0.000000	06			getdents
0.00	0.000000	0	14	14	mkdir
0.00	0.000000	01			chmod
0.00	0.000000	02			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	山羊时间
0.00	0.000000	0	1	set_tid_address
0.00	0.000000	0	1	set_robust_list
<hr/>				
100.00	700.430108		356023	共 212 个评分

## pcap 运行 2

用：

- 160GB pcap 实时 ISP 捕获
- 30k ETPro 规则

注意：这与上面的运行相同。第二次相同的运行的目的是向我们展示是否存在一些小的偏差以及预期这些偏差的位置，以便我们在对修补的内核进行运行时可以牢记这一点。

命令：

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k 无 \
-r /var/log/suricata/pcap/bigtest.pcap --runmode=autofp \&> strace-bigpcap-run-2
```

输出：

```
[13707] 2018 年 9 月 2 日 -- 01: 21: 35 - (conf-yaml-loader.c: 265) <Info>
(ConfYamlPars[13707] 2018 年 9 月 2 日 -- 01: 21: 35 - (suricata.c: 1073) <Notice>
(LogVersion) -- Th[13707] 2018 年 9 月 2 日 -- 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>
(ReceivePcap% timestampsusecs/callcallerrors syscall
```

99.92	646.074279	3032	213119	纳米睡眠
0.07	0.477696	3	139208	203 福泰克斯
0.00	0.014877	30	495	蒙地图
0.00	0.008516	2	5423	BRK
0.00	0.003147	629	5	臀部
0.00	0.001597	3	589	mmap
0.00	0.001145	1769		打开
0.00	0.001144	0	4865	2 阅读
0.00	0.000589	6	100	mprotect 公司
0.00	0.000273	5	60	fstat
0.00	0.000252	3	74	写



0.00	0.000249	4	69	关闭
0.00	0.000230	7	34	34 访问
0.00	0.000177	6	32	克隆
0.00	0.000085	4	22	2 属性
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	山羊时间
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 个 ioctl
0.00	0.000007	7	1	执行
0.00	0.000005	3	2	sysinfo
0.00	0.000005	2	3	prctl
0.00	0.000004	2	2	取消链接
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	插座
0.00	0.000002	2	1	捆
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	听
0.00	0.000000	0	1	setrlimit
<hr/>				
100.00	646.584503		364296	共 257 个评分

## pcap 运行 3

用：

- 8.1GB pcap 实时 ISP 捕获
- 30k ETPro 规则

注意：这与上面的运行相同，但大小不同。第二次运行的目的是向我们展示是否存在一些小偏差以及预期这些偏差的位置，以便我们在对修补的内核进行运行时可以牢记这一点。

命令：

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k 无 \
-r /var/log/suricata/pcap/test.pcap --runmode=autofp \&> strace-smallpcap-run-1
```

输出:

[11774] 2018 年 9 月 2 日 -- 00: 49: 16 - (conf-yaml-loader.c: 265) <Info>  
(ConfYamlPars[11774] 2018 年 9 月 2 日 -- 00: 49: 16 - (suricata.c: 1073) <Notice>  
(LogVersion) -- Th[11774] 2018 年 9 月 2 日 -- 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% timeusecondsusecs/callcallerrors syscall

98.10	33.157150	836	39663	纳米睡眠
1.13	0.381366	5	78731	51 福特斯
0.63	0.212663	9667	22	2 属性
0.06	0.020189	3	6122	BRK
0.02	0.007494	3747	2	取消链接
0.02	0.006948	14	500	蒙地图
0.02	0.006781	1	4865	2 阅读
0.01	0.002132	355	6	臀部
0.00	0.001489	3	589	mmap
0.00	0.000943	14	69	打开
0.00	0.000427	13	32	克隆
0.00	0.000333	3	100	mprotect 公司
0.00	0.000220	4	60	fstat
0.00	0.000158	2	69	关闭
0.00	0.000142	4	34	34 访问
0.00	0.000110	1	74	写
0.00	0.000077	1	74	山羊时间
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 个 ioctl
0.00	0.000009	9	1	捆
0.00	0.000006	23		prctl
0.00	0.000003	13		LSEEK
0.00	0.000003	13		rt_sigprocmask
0.00	0.000002	21		插座
0.00	0.000002	21		听
0.00	0.000002	21		arch_prctl
0.00	0.000001	12		getrlimit
0.00	0.000001	11		set_tid_address
0.00	0.000001	11		set_robust_list
0.00	0.000000	01		执行
0.00	0.000000	01		uname
0.00	0.000000	0	14	14 mkdir
0.00	0.000000	02		sysinfo
0.00	0.000000	01		setrlimit





用：

命令：

输出：

% 时间	秒	usecs/调用	调用	错误	syscall
98.75	28.964359	823	35195		纳米睡眠
1.15	0.336672	4	79472	76	福特克斯
0.05	0.013790	2	6011		BRK
0.02	0.007125	14	507		蒙地图
0.02	0.005256	876	6		臀部
0.00	0.001043	2	589		mmap
0.00	0.000760	0	4865	2	阅读
0.00	0.000349	5	69		打开
0.00	0.000342	3	100		mprotect 公司
0.00	0.000211	7	32		克隆
0.00	0.000173	3	69		关闭
0.00	0.000142	2	60		fstat
0.00	0.000141	4	34	34	访问
0.00	0.000066	5	14	14	mkdir
0.00	0.000058	1	74		写
0.00	0.000037	2	22	2	属性



0.00	0.000015	0	74	山羊时间
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 个 ioctl
0.00	0.000000	0	10	rt_sigaction
0.00	0.000000	0	3	rt_sigprocmask
0.00	0.000000	0	1	插座
0.00	0.000000	0	1	捆
0.00	0.000000	0	1	听
0.00	0.000000	0	1	setsockopt
0.00	0.000000	0	1	执行
0.00	0.000000	0	2	取消链接
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
<hr/>				
100.00	29.330591		127237	共 130 个评分

## 未修补的内核 - 实时运行

使用: - 30k ETPro 规则

我们测量了 Suricata 在 24 小时内运行 3 个不同采样时间的 2 小时。

命令:

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

输出:

```
root@suricata: /home/pevman/tests/meltdown/prepatched-kernel#cat suri.syscalls-1 |grep  
-v ' 0 ' |grep syscalls |排序 -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         系统调用: 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 |排序 -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           系统调用: 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              系统调用: 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 |排序 -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         系统调用: 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
```



## 修补的内核 - 读取 pcap

```
root@suricata: ~# uname -a
Linux suricata 4.15.2-amd64 #1 SMP 2018 年 2 月 8 日星期四 23: 36: 33 CET x86_64 x86_64x86_64
GNU/Linux
```

### pcap 运行 1

用:

- 160GB pcap 实时 ISP 捕获
- 30k ETPro 规则

注意: 这与上面的运行相同。第二次相同的运行的目的是向我们展示是否存在一些小的偏差以及预期这些偏差的位置, 以便我们在对修补的内核进行运行时可以牢记这一点。

命令:

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k 无 \
-r /var/log/suricata/pcap/bigtest.pcap --runmode=autofp \&> strace-bigpcap-run-2
```

输出:

```
[5053] 2018 年 9 月 2 日 -- 11: 07: 24 - (conf-yaml-loader.c: 265) <Info> (ConfYamlParse[5053]
2018 年 9 月 2 日 -- 11: 07: 24 - (suricata.c: 1073) <Notice> (LogVersion) -- Thi[5053] 2018 年
9 月 2 日 -- 11: 08: 05 - (tm-threads.c: 2172) <Notice> (TmThreadWaitOnTh[5053] 2018 年 9 月 2 日
-- 11: 35: 21 - (suricata.c: 2716) <Notice> (SuricataMainLoop) [5076] 2018 年 9 月 2 日 -- 11:
35: 30 - (source-pcap-file.c: 354) <Notice> (ReceivePcap% timesecondssecs/callcallerrors
syscall
```

99.90	749.382002	3641	205839	纳米睡眠
0.10	0.731366	5	143718	113 福特斯
0.00	0.006696	14	495	蒙地图
0.00	0.006093	1	5423	BRK
0.00	0.002849	407	7	臀部
0.00	0.000695	0	4865	2 阅读
0.00	0.000531	1	589	mmap
0.00	0.000288	3	100	mprotect 公司
0.00	0.000208	7	32	克隆
0.00	0.000173	2	74	写
0.00	0.000161	2	69	打开



0.00	0.000101	3	34	34 访问
0.00	0.000093	2	60	fstat
0.00	0.000090	1	69	关闭
0.00	0.000021	1	22	2 属性
0.00	0.000008	4	2	取消链接
0.00	0.000008	0	74	山羊时间
0.00	0.000005	1	10	rt_sigaction
0.00	0.000005	2	3	rt_sigprocmask
0.00	0.000005	5	1	捆
0.00	0.000003	3	1	插座
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 个 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	听
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	执行
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	共 167 个评分

## pcap 运行 2

用：

- 160GB pcap 实时 ISP 捕获
- 30k ETPro 规则

注意：这与上面的运行相同。第二次相同的运行的目的是向我们展示是否存在一些小的偏差以及预期这些偏差的位置，以便我们在对修补的内核进行运行时可以牢记这一点。

命令：

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \
--pidfile /var/run/suricata.pid -l /tmplog/ -k 无 \
-r /var/log/suricata/pcap/bigtest.pcap --runmode=autofp \&> strace-bigpcap-run-2
```









0.00	0.000033	33	1	插座
0.00	0.000029	10	3	rt_sigprocmask
0.00	0.000026	0	69	关闭
0.00	0.000020	3	6	getdents
0.00	0.000012	6	2	2 个 ioctl
0.00	0.000008	3	3	LSEEK
0.00	0.000006	6	1	捆
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	听
0.00	0.000000	0	34	34 访问
0.00	0.000000	0	1	执行
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
<hr/>				
100.00	37.248372		142449	共 134 个评分

## pcap 运行 4

用：

- 8.1GB pcap 实时 ISP 捕获
- 30k ETPro 规则

注意：这与上面的运行相同。第二次相同的运行的目的是向我们展示是否存在一些小的偏差以及预期这些偏差的位置，以便我们在对修补的内核进行运行时可以牢记这一点。

命令：

```
strace -c /usr/local/bin/suricata -c /etc/suricata/suricata.yaml \  
--pidfile /var/run/suricata.pid -l /tmplog/ -k 无 \  
-r /var/log/suricata/pcap/test.pcap --runmode=autofp \&> strace-smallpcap-run-2
```

输出：

```
[4855] 2018 年 9 月 2 日 -- 11: 05: 45 - (conf-yaml-loader.c: 265) <Info> (ConfYamlParse[4855]  
2018 年 9 月 2 日 -- 11: 05: 45 - (suricata.c: 1073) <Notice> (LogVersion) -- Thi[4855] 2018 年  
9 月 2 日 -- 11: 06: 39 - (tm-threads.c: 2172) <Notice> (TmThreadWaitOnTh
```





[4855] 2018 年 9 月 2 日 -- 11: 07: 17 - (suricata.c: 2716) <Notice> (SuricataMainLoop)  
 [4918] 2018 年 9 月 2 日 -- 11: 07: 23 - (source-pcap-file.c: 354) <Notice>  
 (ReceivePcap% timesecondsusecs/callcallerrors syscall

98.33	26.938375	667	40380	纳米睡眠
1.59	0.435495	5	81020	170 福特克斯
0.04	0.010092	2	5923	BRK
0.02	0.005007	10	507	蒙地图
0.02	0.004506	644	7	臀部
0.00	0.000815	1	589	mmap
0.00	0.000676	0	4865	2 阅读
0.00	0.000287	9	32	克隆
0.00	0.000284	3	100	mprotect 公司
0.00	0.000189	3	74	写
0.00	0.000083	1	69	打开
0.00	0.000068	1	74	山羊时间
0.00	0.000058	1	69	关闭
0.00	0.000055	1	60	fstat
0.00	0.000044	1	34	34 访问
0.00	0.000025	3	10	rt_sigaction
0.00	0.000018	6	3	rt_sigprocmask
0.00	0.000016	1	22	2 属性
0.00	0.000012	2	6	getdents
0.00	0.000009	9	1	听
0.00	0.000007	2	3	LSEEK
0.00	0.000006	3	2	2 个 ioctl
0.00	0.000005	3	2	取消链接
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	插座
0.00	0.000003	3	1	捆
0.00	0.000003	31		set_tid_address
0.00	0.000003	31		set_robust_list
0.00	0.000001	11		chmod
0.00	0.000000	01		setsockopt
0.00	0.000000	01		执行
0.00	0.000000	01		uname
0.00	0.000000	0	14	14 mkdir
0.00	0.000000	02		sysinfo
0.00	0.000000	01		setrlimit
100.00	27.396158		133883	共 224 个评分





## 修补的内核 - 实时运行

使用：- 30k ETPro 规则

我们测量了 Suricata 在 24 小时内运行 3 个不同的采样时间，这次是在修补的内核上。

命令：

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

输出：

```
root@suricata: /home/pevman/tests/meltdown/postpatched-kernel#cat suri.syscalls-1  
|grep -v ' 0'|grep syscalls |排序 -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	系统调用: 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 |排序 -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	系统调用: 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 |排序 -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        系统调用: 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

```



## 观察

在上面关于 strace 的结果中，我们在第一种情况下使用 cluster-type: cluster\_flow 在常规 af-packet 模式下运行 Suri-cata，第二次运行（使用修补的内核），我们使用 XDP andaf-packet cluster-type: cluster\_qm 运行 Suricata。使用 syscalls: sys\_enter\_bpf 很明显。

总体而言，根据完成的测试，没有明确观察到任何性能损失（尤其是在额外 5-25% CPU 使用率的范围内）。也没有观察到 Suricata 的一般性能对丢包/memcap 命中或更高的 CPU 使用率的任何不利影响。

这些观察结果已在 Mozilla 的生产中得到证实。从发布之日起，我们就一直在运行 Meltdown 修补内核，我们没有看到明显的性能下降。

## 技术细节

Meltdown 缓解措施（它不会解决问题，它使它无法滥用）是 KPTI（适用于 Linux）和 KvaShadow（适用于 Windows）。这种缓解措施的作用是从用户空间中删除大部分内核映射（用户空间映射在



kernel 页表仍然存在)。

现在, 大多数这些映射都已消失, 每个系统调用都需要重新加载 full 或 partialpage 表, 从 TLB 刷新部分或全部数据 (以及来自指令 TLB 的部分或全部数据, 这是独立的)。

为了分摊成本影响, 可以将 PCID (进程上下文标识符) 用于具有 PCID 的 CPU。最初被认为是 VMEXIT 情况的性能优化, 当虚拟机管理程序在虚拟机之间切换时 (因此经常丢弃 TLB) 现在可以重新用于用户空间 -> 内核空间切换。

虽然 PCID 存在于 Intel 平台上, 从 Sandy Bridge 开始, 只有从 Haswellup 开始, 它们具有 INVPCID, 可以精确地从 TLB 中删除单个条目, 而不是刷新整个条目。

使用 PCID+INVPCID 时, KPTI 的性能影响应该不会很明显。如果可以的话, 你想运行 Haswell。

就其他缓解措施而言, IBRS + IBPB 可能会对 Skylake 之前的所有内容产生明显的性能影响。我们没有对其进行测试, 因为没有稳定的 CPU 微码可用。

鉴于所有这些以及我们对这些攻击的了解, 人们可能会想知道.....

## 我应该担心吗?

在其中, 我们带领读者穿越一个美丽的风险管理世界。Microsoft 发布了一个与这些漏洞相关的风险管理的精彩示例

我们强烈建议您阅读它, 并花点时间考虑一下这里的场景。根据设计, NSM 传感器应该位于一个隔离的主机上, 该主机不暴露任何接口, 但最少的接口。应该发送日志, 唯一的服务应该是带有 MFA 的 SSH (来自堡垒主机之类的东西)。不应运行不必要的服务。

我们的读者是否应该担心高度科学且不太可能的攻击, 这些攻击要求威胁行为者位于同一主机上? 如果威胁行为者已经拥有本地 shell, 他们是否可以通过其他方式完成他们想要的事情?

尽管如此, Linux 本地权限提升历史记录看起来是可利用的, 因此可以安全地假设有许多比在本地主机上使用 Meltdown 和 Spectre 攻击更舒适的方法来提升本地权限。

## 但您建议使用 aSpectre PoC 中使用的 eBPF

是的, 我们有。细心的读者会发现, 使用 eBPF 只是重新搜索所做的优化, 他们还有其他几种方法可以在 SpectrePoC 中找到要滥用的小工具。他们只是以简短的 eBPF 程序的形式带来了小工具, 以证明这一点。



eBPF 代码已得到显著强化。研究人员用于绕过验证的方法（通过调用未验证提供的 eBPF 代码的函数）已被删除。JIT 常数盲法无处不在。

顺便说一句，KVM 通过在 VMEXIT 上清除寄存器在 5 秒内打破了 PoC:)虽然 eBPF JIT 中肯定还有其他漏洞，但我们建议对使用这些漏洞的可能性进行风险评估。







## 第 4 章

# 延伸阅读

XDP:

[1] [http://people.netfilter.org/hawk/presentations/LLC2017/XDP\\_DDoS\\_protecting\\_LLC2017.pdf](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](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](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](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](https://people.netfilter.org/hawk/presentations/NetDev2.2_2017/XDP_for_the_Rest_of_Us_Part_2.pdf)

Netronome 白皮书[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 系统调用: [13] <http://www.brendangregg.com/perf.html>





## 第 5 章

# 作者

在暴民中，我们信任  
关于作者：

米哈尔·普尔津斯基 (@MichalPurzynski)

- 威胁管理，Mozilla
- 入侵检测
- 数字取证
- 事件响应

彼得·马涅夫 (Peter Manev) (@pevma)

- Suricata 核心团队
- 首席 QA 和培训讲师
- Stamus 网络
- Mobster 布道者





## 第 6 章

# 谢谢

没有他们，本指南就不可能完成：

- Eric Leblond (@regiteric – Suricata AFP/XDP 神一般的开发人员正在做内核补丁同时从键盘上赶猫)
- Jesper Brouer (@netoptimizer, RedHat XDP 开发首席内核工程师歌剧)
- Dave Miller for AFPPacket : -)
- IOvisor 项目
- SuriCon 2017 ! !
- Suricata 社区激励我们进一步突破极限

