Closing the BPF map permission loophole

Lorenz Bauer <i@lmb.io>
Maintainer github.com/cilium/ebpf
Contents

● Origin story
● Access control of BPF maps
● Break things
Background

- Used to work for Cloudflare, all things BPF
- [github.com/cloudflare/tubular](https://github.com/cloudflare/tubular): a CLI for “BSD sockets on steroids”
  - Listen on all ports on an IP address!
- Built on sk_lookup mentioned by Martin Lau yesterday
Give read-only access to unprivileged users

$ tubectl status
opened dispatcher at /sys/fs/bpf/4026531840_dispatcher

Bindings:
protocol prefix port label
  tcp 127.0.0.0/8 0   foo

Destinations:
label domain protocol socket lookups misses errors
  foo   ipv4   tcp   sk:-   0   0   0
Tubular stores state in /sys/fs/bpf

$ ls -l /sys/fs/bpf/4026531840_dispatcher
total 0
-rw-r----- 1 tubular tubular 0 Aug 23 14:40 bindings
-rw-r----- 1 tubular tubular 0 Aug 23 14:40 destination_metrics
-rw-r----- 1 tubular tubular 0 Aug 23 14:40 destinations
-rw-r----- 1 tubular tubular 0 Aug 23 14:40 link
-rw-r----- 1 tubular tubular 0 Aug 23 14:40 program
-rw-r----- 1 tubular tubular 0 Aug 23 14:40 sockets
Read-only access via BPF_OBJ_GET

```
BPF_OBJ_GET(/sys/fs/bpf/.../bindings, BPF_F_RDONLY) = fd
```
Ways to restrict modifications of BPF maps

<table>
<thead>
<tr>
<th></th>
<th>From syscall</th>
<th>From BPF program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per map</td>
<td><code>bpf(BPF_MAP_FREEZE)</code></td>
<td><code>BPF_F_RDONLY_PROG, ...</code></td>
</tr>
<tr>
<td>Per fd</td>
<td></td>
<td><code>BPF_F_RDONLY, ...</code></td>
</tr>
<tr>
<td>Per pinned file</td>
<td><code>chmod(2)</code></td>
<td>N/A</td>
</tr>
</tbody>
</table>

This slide contains a lie.
Where are permissions kept?

<table>
<thead>
<tr>
<th></th>
<th>From syscall</th>
<th>From BPF program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per map</td>
<td>struct bpf_map-&gt;frozen</td>
<td>struct bpf_map-&gt;map_flags</td>
</tr>
<tr>
<td>Per fd</td>
<td>struct fd-&gt;f_mode</td>
<td></td>
</tr>
<tr>
<td>Per pinned file</td>
<td>struct inode-&gt;i_mode</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Given a read-only map fd, ...
Given a read-only map $fd$, ...
1. it’s not possible to modify the map
Given a read-only map fd, ...
1. it’s not possible to modify the map
2. it’s not possible to obtain a read-write fd
Read-only map fds can be modified via BPF program

1. Take a read-only map fd
2. Craft a BPF program that calls bpf_map_update_elem(read-only fd)
3. Load the program
4. Execute the program (PROG_RUN, etc.)
<table>
<thead>
<tr>
<th></th>
<th>From syscall</th>
<th>From BPF program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per map</td>
<td><code>struct bpf_map-&gt;frozen</code></td>
<td><code>struct bpf_map-&gt;map_flags</code></td>
</tr>
<tr>
<td>Per fd</td>
<td><code>struct fd-&gt;f_mode</code></td>
<td><code>NOPE</code></td>
</tr>
<tr>
<td>Per pinned file</td>
<td><code>struct inode-&gt;i_mode</code></td>
<td>N/A</td>
</tr>
</tbody>
</table>

This slide contains no more lies.
Fix #1: Refuse map fd which is not read-write

Pro:

- Very simple
- Backportable?
- High risk of breaking users
  - However, test_progs and test_maps are happy

Con:

- BPF programs that only read are rejected
Fix #2: Track map permissions using bpf_type_flag

--- a/include/linux/bpf.h
+++ b/include/linux/bpf.h
@@ -397,6 +397,9 @@ enum bpf_type_flag {

    /* DYNPTR points to a ringbuf record. */
    DYNPTR_TYPE_RINGBUF = BIT(9 + BPF_BASE_TYPE_BITS),
+    /* MEM is write-only. Used with map values. */
+    MEM_WRONLY = BIT(10 + BPF_BASE_TYPE_BITS),

    __BPF_TYPE_FLAG_MAX,
    __BPF_TYPE_LAST_FLAG = __BPF_TYPE_FLAG_MAX - 1,
Fix #2: Store bpf_type_flag in bpf_reg_type

dst_reg->type = PTR_TO_MAP_VALUE | MEM_RDONLY;
dst_reg->type = PTR_TO_MAP_VALUE | MEM_WRONLY;
dst_reg->type = PTR_TO_MAP_VALUE;

Fix #2: Track map permissions using bpf_type_flag

Pro:
- Less likely to break users
- BPF programs that only read are accepted

Con:
- Definitely no backport
- Requires auditing PTR_TO_MAP_VALUE, possibly others
- I don’t trust myself to pull this off without help
Opinions?
Given a read-only map `fd`, …
1. it’s not possible to modify the map
2. it’s not possible to obtain a read-write `fd`
Read-only map fds can be made read-write

1. Take a read-only map fd
2. BPF_OBJ_PIN into /sys/fs/bpf
3. Open pinned map with open_flags == 0
Reason: BPF_OBJC_PIN doesn’t check fd permissions

- It’s possible to pin a read-only fd
- Pinned inode is always owned by current user
- Pinned inode always has o+rw permissions

NB: same problem applies to pinned programs and links.
Fix #1: enforce that fd is R/W in BPF_OBJ_PIN

Pro:

- Simple
- test_progs and test_maps are happy

Con:

- It’s impossible to pin a map created with BPF_F_RDONLY, BPF_F_WRONLY
  - Pin R/W + chmod() still possible though
Fix #2: adjust permissions + prevent chmod() escalation

- In BPF_OBJ_PIN, adjust created file permissions to match `fd->f_mode`
  - Read-only fd leads to o=r file instead of o=rw
- In chmod(2), prevent raising permissions
  - From o=rw to o=r / o=w is OK
  - From o=r to o=rw / o=w is not OK
Fix #2: adjust permissions + prevent chmod() escalation

Pro:

● Allows pinning BPF_F_RDONLY, … fds
● Probably less likely to break user space

Con:

● Somewhat weird chmod semantics
● Other ways to change file mode?
More opinions?
Thanks!