

> Martin Schmidt

# Performing open-syscall surgery on SSHd with eBPF as a scalpel

Martin Schmidt

EPITA

November 06, 2021



# What is eBPF

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

#### eBPF – *ebpf.io*

eBPF is a revolutionary technology with origins in the Linux kernel that can run sandboxed programs in an operating system kernel. It is used to safely and efficiently extend the capabilities of the kernel without requiring to change kernel source code or load kernel modules.

#### Tracing with eBPF

eBPF is used a lot for tracing and monitoring

syscalls

internal kernel functions

user program functions

hardware events:

■ cache-misses, branch-misses, ...

and more...



How do I get started tracing with eBPF ?

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

- You could write eBPF bytecode by hand
- Could compile your eBPF script with linux in-tree (uses clang -target bpf)



# Using front-ends for eBPF

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

The main and recommended front-ends for BPF tracing are BCC and bpftrace

#### bpftrace

- for one-liners and short scripts.
- awk-like syntax, quick and easy

#### bcc

- for complex tools and daemons
- write the eBPF program (the probes) in C
- Have a piece of python receiving events from probes
   We'll look at some examples using bpftrace

> Martin Schmidt

#### Definition

uprobe instruments the beginning of a user-level function's execution, and uretprobe instruments the end (its return).

#### Example: tracing readline in bash

```
uretprobe:/bin/bash:readline
```

```
time("%H:%M:%S ");
printf("%-6d %s\n", pid, str(retval));
```

# kprobe/kretprobe

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

### Example: counting calls to vfs related functions

#### kprobe:vfs\_\*

```
@[func] = count();
```



# Example: Trace tcp connect()s with kprobe

kprobe:tcp\_connect

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

> > Martin Schmidt (EPITA)

```
$sk = ((struct sock *) arg0);
$inet_family = $sk->__sk_common.skc_family;
```

```
if ($inet_family == AF_INET) {
   $daddr = ntop($sk->._sk_common.skc_daddr);
   $saddr = ntop($sk->._sk_common.skc_rcv_saddr);
   $lport = $sk->._sk_common.skc_num;
   $dport = $sk->._sk_common.skc_dport;
   $dport = ($dport >> 8)
        | (($dport << 8) & 0x00FF00);
   }
}</pre>
```



### tracepoint

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

#### Example: tracing calls to the openat syscall

```
tracepoint:syscalls:sys_enter_openat
```

```
/comm == "cat"/
```

```
printf("%s %s\n", comm, str(args->filename));
```



> Martin Schmidt

Portability

- How does the previous example work ?
- How do we have the parameter's type information ?



### BTF to the rescue

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

#### BTF (BPF Type Format)

BPF Type Format, which provides struct information to avoid needing Clang and kernel headers.

name: sys\_enter\_openat ID: 645 format: field:unsigned short common\_type; field:unsigned char common\_flags; field:unsigned\_char\_common\_preempt\_count; offset:3; field:int common\_pid; offset:4; size:4: signed:1: size:4: signed:1: field:int dfd: offset:16: size:8: signed:0: field:const char attribute ((user)) \* filename: size:8: signed:0: field:int flags: offset:32: size:8: sianed:0: field:umode\_t\_mode; offset:40: size:8: signed:0:

Figure: /sys/kernel/debug/tracing/events/syscalls/sys\_enter\_openat/format



> Martin Schmidt

### Now what ?

Lots of cool examples but can we do anything cool/useful ?

#### My projects:

tracing sshd

- ebpf firewall
- maybe on-the-fly decryption in the future :eyes:



> Martin Schmidt

The subject

- Playing around with cool looking examples is fun but...
  - We wanted something useful
- Try the observability part of eBPF by tracing a program.



The target: *OpenSSH's* sshd

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

- The target: *OpenSSH's* sshd
- Especially sshd AuthorizedKeysCommand

#### AuthorizedKeysCommand

Specifies a program to be used to look up the user's public keys. The program must be owned by root, not writable by group or others and specified by an absolute path. Arguments to AuthorizedKeysCommand accept the tokens described in the TOKENS section. If no arguments are specified then the username of the target user is used.



> Martin Schmidt

# Why is this needed ?

 There exists infrastructures where the AuthorizedKeysCommand program is a bottleneck and people want to log the time it took to execute.

> Martin Schmidt

> > }

# First try (eBPF uprobes)

#### uprobe:/usr/bin/sshd:do\_authenticated

printf("[do\_authenticated (tid: %lu)]\n", tid);



### First try (with symbols)

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

- I just added symbols to my sshd, restarted
- Started looking at the functions in sshd I would want to trace

#### Found some interesting functions

- do\_authenticated
  - Takes care of post-authentication
- do\_cleanup
  - Cleans up once user session is finished
- user\_key\_allowed
  - Checks if a user key is allowed (runs the AuthorizedKeysCommand)

LSE

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

### First try (symbols): what does a probe look like ?

```
auths.insert(&pid, &auth);
```

```
return 0;
```

Martin Schmidt (EPITA)



# First try (symbols)

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt







| TIME (ms)



First try (with\_symbols): conclusion

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

It works very well and can be extended but...

#### Shortcomings

- I have included headers directly from sshd's sources in order to have parameter types
- We can't rebuild with symbols on the infrastructure being debugged

scalpel Martin Schmidt

#### Life sucks

Second try

■ We can't have symbols ;(



# Second try

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

#### Solution

Trace the syscalls

#### We can't just stupidly trace all execve syscalls

- Lots of noise from the user session (/bin/sh, ...)
- Log more stuff than just programs being executed



### Second try: settings goals

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

#### Goals

#### authentications

login of user

- time & date
- if it succeeded

#### ssh sessions

- start time
- end time
- commands that were ran inside
  - arguments to the command
  - return value
  - duration



> Martin Schmidt

# Second try: getting the thing to work

I started by straceing sshd hoping I would see a magical and useful pattern



> Martin Schmidt

# Second try: getting the thing to work

I had to dive into the source code

#### At first

- I thought *privsep* was a special OpenBSD feature
  - So I ignored the related code...
  - And failed miserably...



# Second try (without privsep)

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt



#### Figure: Flow without privsep

> Martin Schmidt

# Second try (without privsep)

#### Screenshot

```
sshd_authorizedkeyscommand_ran
    {username="martin", duration_ms=1010.226717}
sshd_authorizedkeyscommand_ran
    {username="martin", duration_ms=1007.266651}
sshd_auth_finished
    {username="martin", success=1}
```



Second try (with privsep)

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

- I recompiled with privsep enabled
- After careful reading and drawing...



### Second try: flow analyzed

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt



Figure: sshd auth flow



Second try: the result

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

#### DEMO TIME

■ to the gods of conferencs, plz work

### Second try: conclusion

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

Works pretty well, I'm happy. *Still works 1.5 years later even though a bit hacky* 

#### Shortcomings

Can't read all of argv because argc is unbounded

#### Getting the user

- Depends on the AuthoziedKeysCommand config
- If of the form: <program> (no arguments) user will be
  argv[1]
  - Easy we know where it is



### l'm done

Performing open-syscall surgery on SSHd with eBPF as a scalpel

> Martin Schmidt

#### Questions ?