Kernel Runtime Security Instrumentation

KP Singh

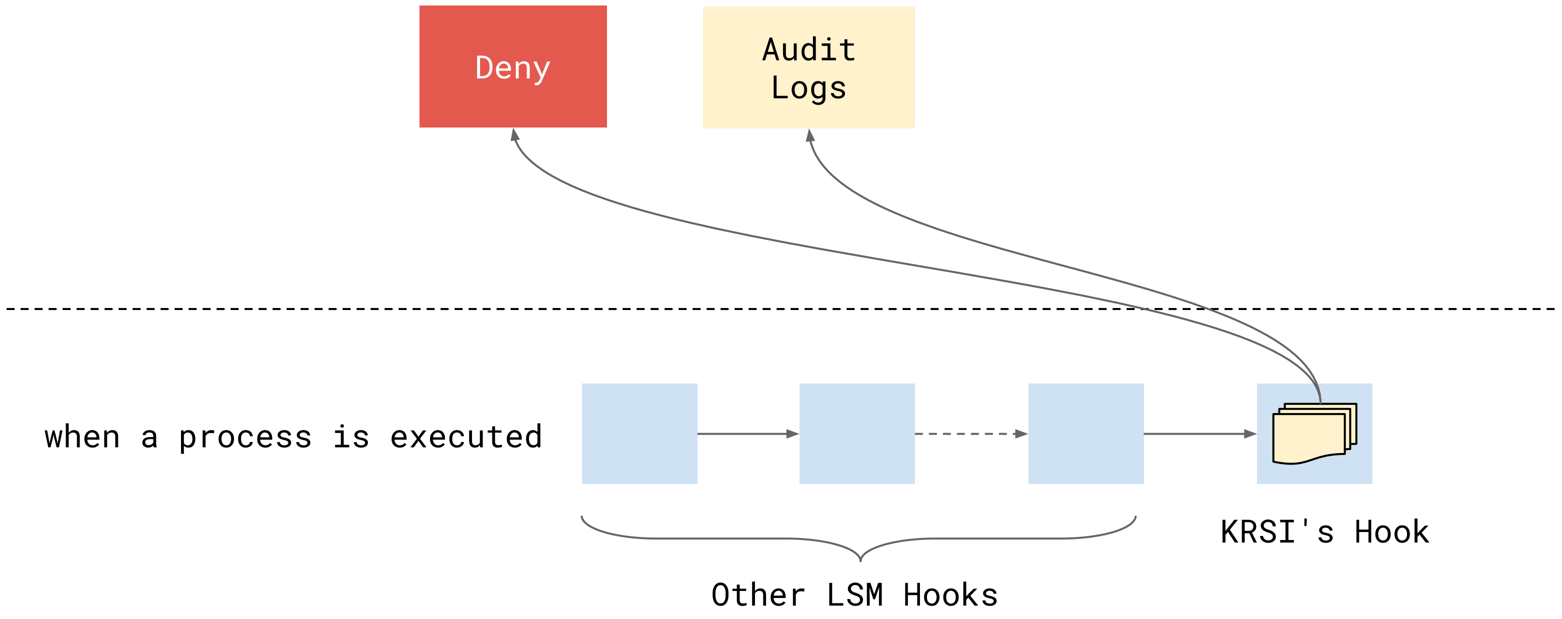
What?

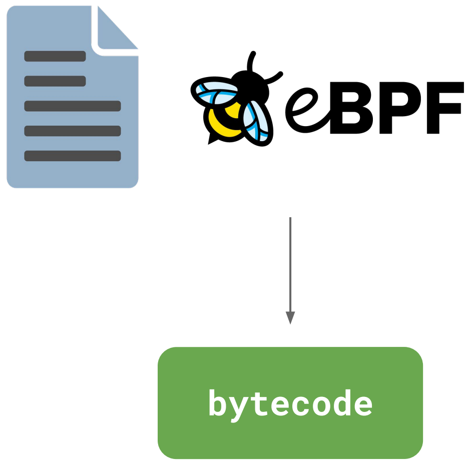
eBPF + LSM

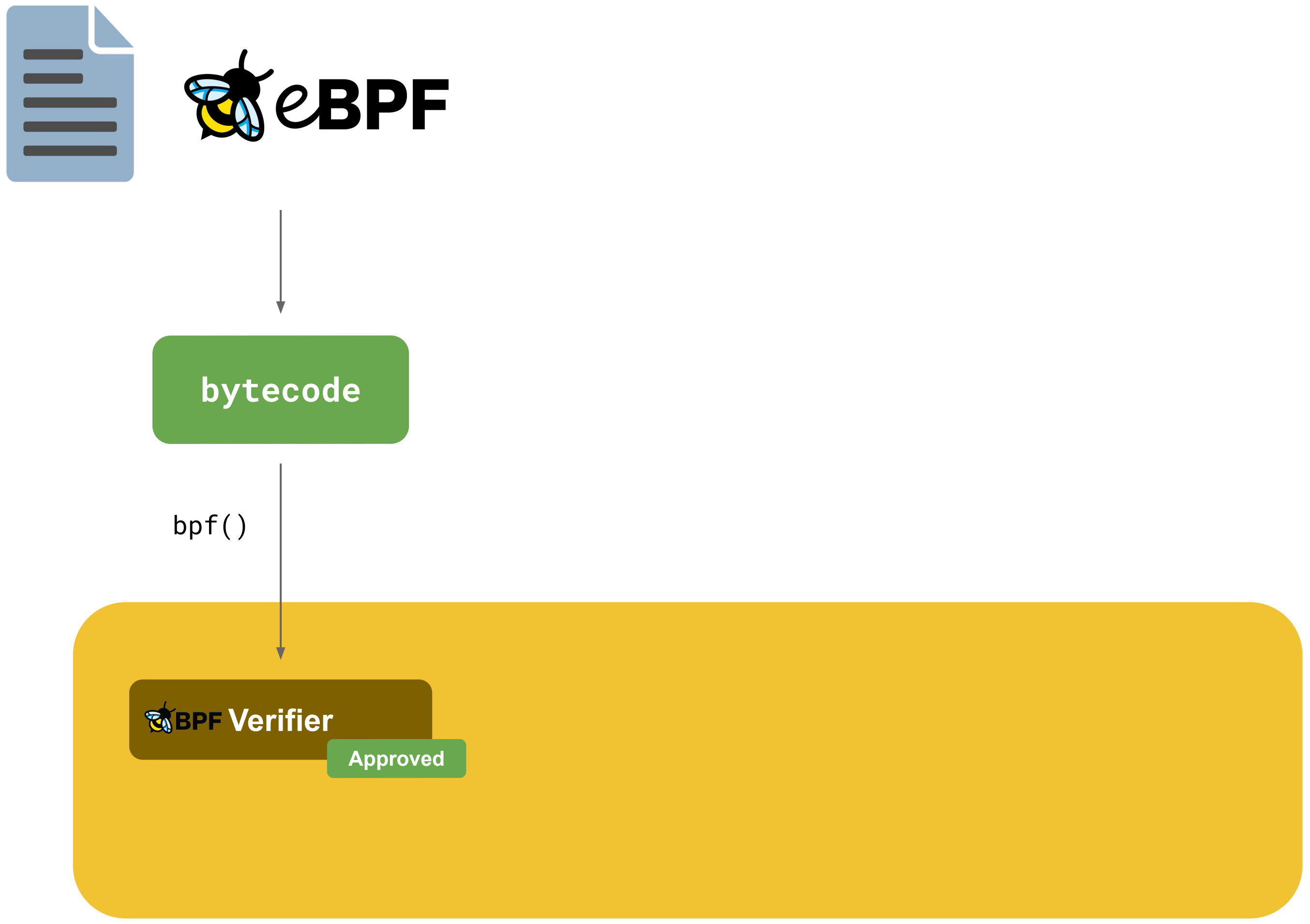
=

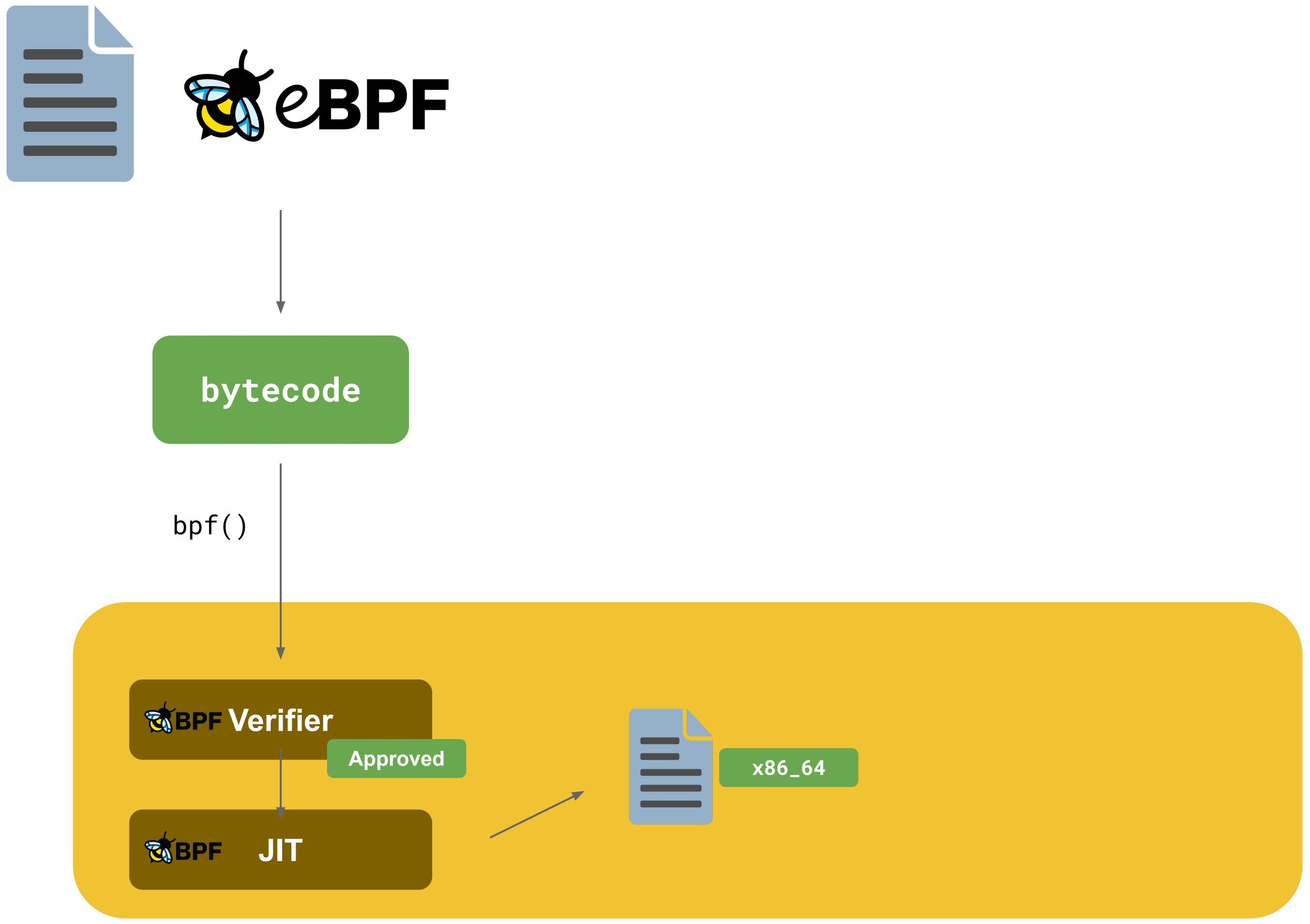
KRSI

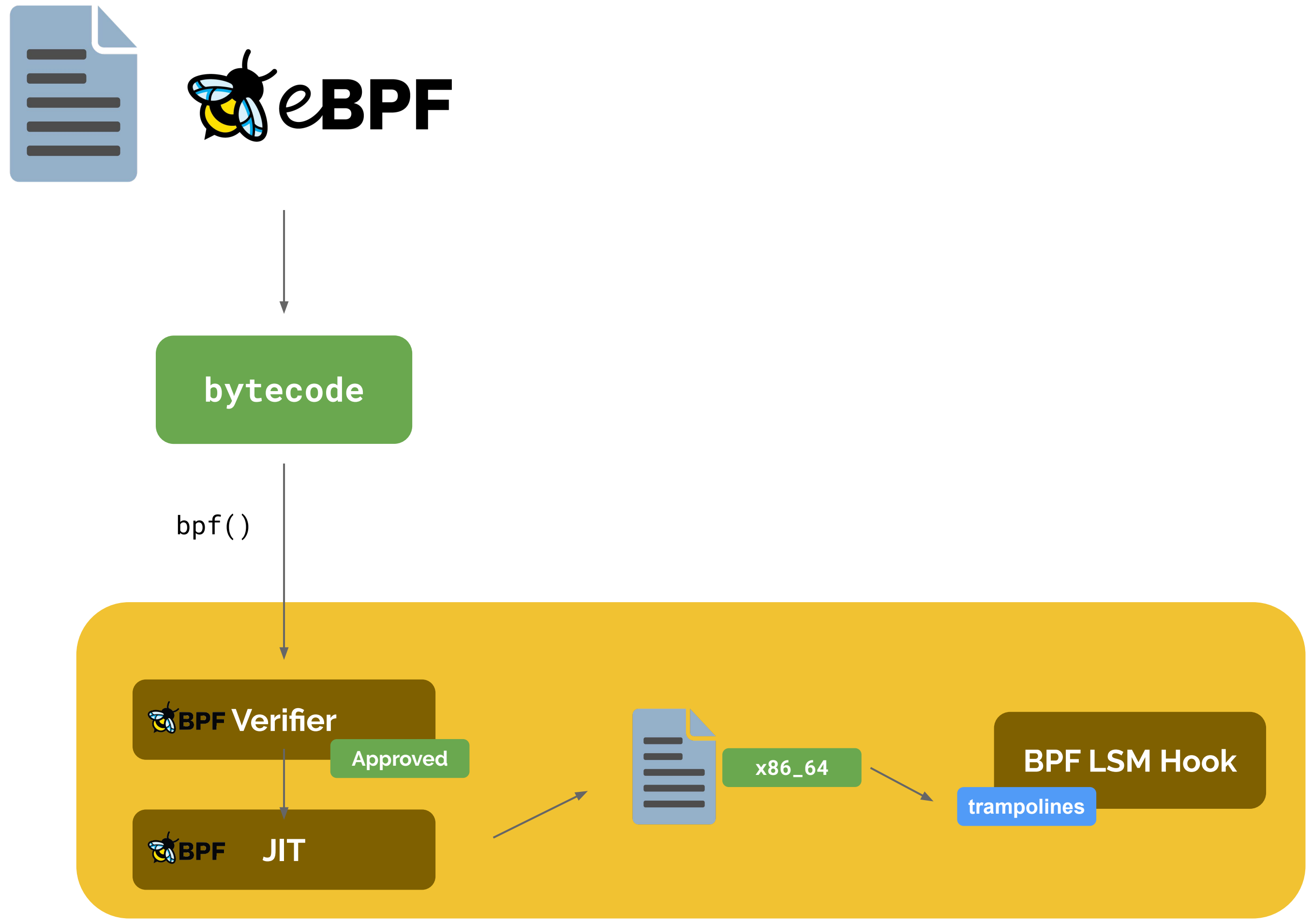
Audit + MAC


Deny
Audit Logs
when a process is executed
KRSI's Hook
Other LSM Hooks


eBPF
bytecode


eBPF
bytecode
bpf()
BPF Verifier
Approved


eBPF
bytecode
bpf()
BPF Verifier
Approved
BPF JIT
x86_64


eBPF
bytecode
bpf()
BPF Verifier
Approved
BPF JIT
x86_64
BPF LSM Hook
trampolines

BPF Trampolines: Types

```
                                 int bpf_lsm_bprm_check_security
fentry         →                 {

fmod_ret       →                   <update ret by calling fmod_ret progs>

                                   if (ret != 0)
                                     goto fexit;

                                 original_function:

                                   <side_effects_happen_here>

fexit          →                 }
```

BPF Trampolines: Types

$$\begin{tikzcd}
& \texttt{BPF\_PROG\_TYPE\_LSM} \arrow[dl, "\texttt{void}\text{ Hooks}"'] \arrow[dr, "\texttt{int}\text{ Hooks}"] & \\
\texttt{BPF\_TRACE\_FEXIT} & & \texttt{BPF\_MODIFY\_RETURN}
\end{tikzcd}$$

LSM Hooks

LSM_HOOK Macros (`lsm_hook_defs.h`)

```
LSM_HOOK(
    int,                            // Return Type
    0,                              // Default Return Value
    bprm_check_security,            // Name
    struct linux_binprm *bprm       // Parameters
)
```

Google

"Macro magic"

Default Callbacks

```
#define LSM_HOOK(RET, DEFAULT, NAME, ...)          \
                                                    \
  noinline RET bpf_lsm_##NAME(__VA_ARGS__)          \
  {                                                 \
    return DEFAULT;                                 \
  }

#include <linux/lsm_hook_defs.h>

#undef LSM_HOOK
```

Google

Default Callbacks

```
[...]

noinline int
bpf_lsm_bprm_check_security(struct linux_binprm *bprm)
{
  return 0;
}


[...]
```

Initialize LSM Hooks

```
#define LSM_HOOK(RET, DEFAULT, NAME, ...) \
  LSM_HOOK_INIT(NAME, bpf_lsm_##NAME),
  #include linux/lsm_hook_defs.h>

#undef LSM_HOOK
```

Initialize BPF LSM Hooks

```
[...]

LSM_HOOK_INIT(bprm_check_security,
              bpf_lsm_bprm_check_security);

[...]
```

Implementing Hooks

Load a program for `bprm_check_security`

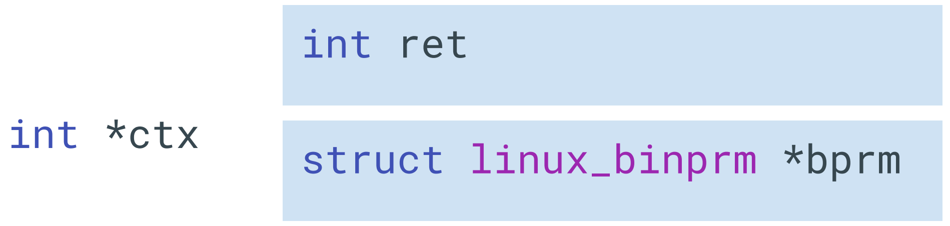
BPF Program

```
SEC("lsm/bprm_check_security")
int BPF_PROG(my_prog, struct linux_binprm *bprm, int ret)
{
    __u32 pid = bpf_get_current_pid_tgid() >> 32;

    if (monitored_pid == pid)
        bprm_count++;

    return 0;
}
```

Context Simplification



```
int *ctx

int ret
struct linux_binprm *bprm
```

```
BPF_PROG(my_prog, struct linux_binprm *bprm, int ret)
```

```
my_prog(int *ctx) {
  __my_prog(ctx[0], ctx[1])
}
```

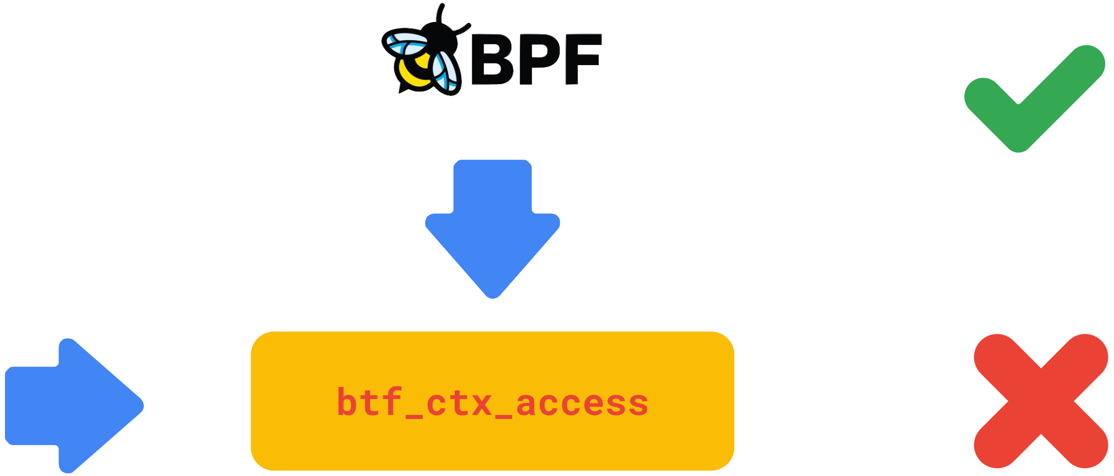
Google

Verification

`/sys/kernel/btf/vmlinux`

Compact type information (BTF)

~125MB of DWARF



Verifier

BPF LSM Hooks: Object File

```
$ objdump -Sr kernel/bpf/bpf_lsm.o

LSM_HOOK(int, 0, bprm_check_security, struct linux_binprm *bprm)

100:   e8 00 00 00 00          callq  105 <bpf_lsm_bprm_check_security+0x5>
101:                           R_X86_64_PLT32 __fentry__-0x4
105:   31 c0                   xor    %eax,%eax>:
107:   c3                      retq
108:   0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
```

BPF LSM Hooks: after `__init`

```
LSM_HOOK(int, 0, bprm_check_security, struct linux_binprm *bprm)

100:    0f 1f 44 00 00          nopl 0x00(%eax,%eax,1)

105:    31 c0                   xor    %eax,%eax>:
107:    c3                      retq
108:    0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
```

ftrace_nop_initialize

BPF Trampoline Update

```
LSM_HOOK(int, 0, bprm_check_security, struct linux_binprm *bprm)

100:    e8 00 00 00 00 64        callq  <trampoline_image>

105:    31 c0                    xor    %eax,%eax>:
107:    c3                       retq
108:    0f 1f 84 00 00 00 00     nopl   0x0(%rax,%rax,1)

200: <trampoline_image>                         arch_prepare_bpf_trampoline
```

LSM Trampolines: Creation

```
push    %rbp
mov     %rsp,%rbp
sub     $0x10,%rsp
push    %rbx
```

Create a frame for a stack size of 16 (`0x10`) bytes:

- 8 bytes for `struct linux_binprm *bprm`
- 8 bytes to save the return value

LSM Trampolines: Invocation

```
mov     %rdi,-0x10(%rbp)

xor     %eax,%eax
mov     %rax,-0x8(%rbp)

callq   __bpf_prog_enter
mov     %rax,%rbx

lea     -0x10(%rbp),%rdi

callq   addr_of_jited_lsm_prog
mov     %rax,-0x8(%rbp)

movabs  $addr_struct_bpf_prog,%rdi
mov     %rbx,%rsi
callq   __bpf_prog_exit
```

} Save the first argument on the stack

} Clear out the return value passed to first LSM program.

} ctx (`int` * array) for the BPF program

} Call the JITed program
Save the return value on the stack

Google

LSM Trampolines: BPF_MODIFY_RETURN

```
  cmpq    $0x0,-0x8(%rbp)
  jne     <do_exit>
```

Skip calling the original function and the rest of the programs upon a non-zero return value

```
 [...]

  mov     -0x10(%rbp),%rdi
  callq   <bpf_lsm_bprm_check_security+0x5>
  mov     %rax,-0x8(%rbp)

  nopl    0x0(%rax,%rax,1)
  nopw    0x0(%rax,%rax,1)
```

nops to align jump target

```
do_exit:
```

BPF Trampolines - Exit

```
mov     -0x8(%rbp),%rax
```

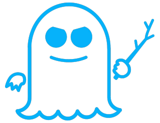
$\}$ Update the return value from the stack

```
pop     %rbx
leaveq
add     $0x8,%rsp
retq
```

Improvements

Indirect Calls

```
hlist_for_each_entry(P, &security_hook_heads.FUNC, list)   \
{                                                          \
    RC = P->hook.FUNC(__VA_ARGS__);                        \
    if (RC != 0)                                           \
        break;                                             \
}
```



Indirect calls worsened by retpolines

and default callbacks are added everywhere!!

So How bad is it?

```c
int main(void) {
    int fd = eventfd(0, 0);
    int c = 10000;

    while (c--)
        eventfd_write(fd, 1);

    return 0;
}
```

```c
int main(void) {
    int fd = eventfd(0, 0);
    int c = 10000;

    while (c--)
        eventfd_write(fd, 1);

    return 0;
}
```

```
int main(void) {
    int fd = eventfd(0, 0);
    int c = 10000;

    while (c--)
        eventfd_write(fd, 1);

    return 0;
}
```

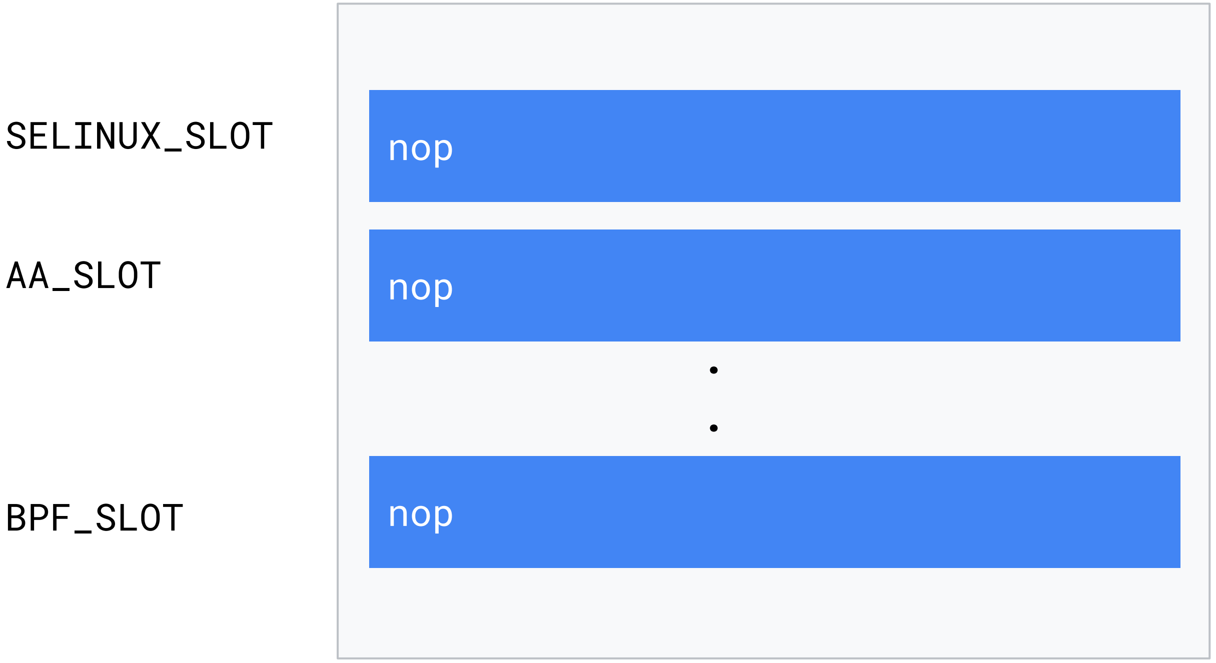
+ ~4%

Google

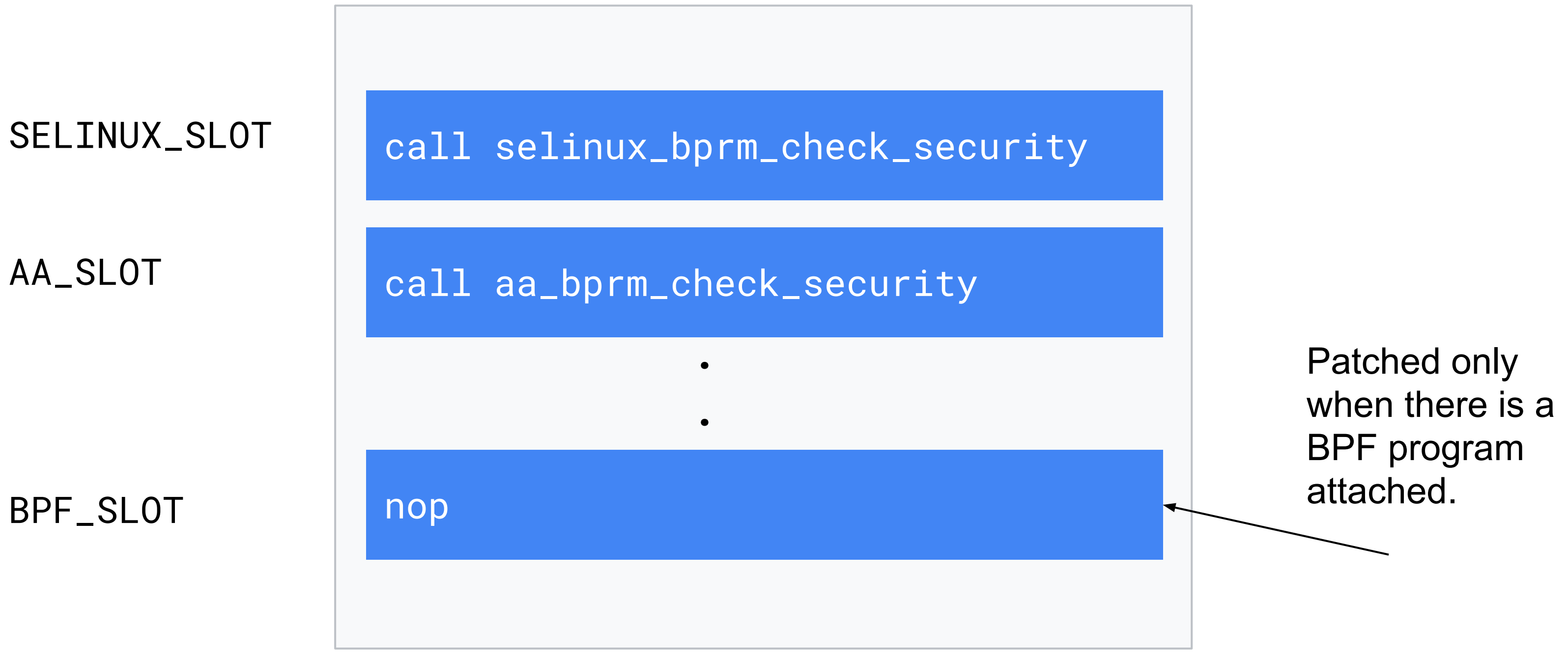
We know the addresses of LSM Hooks at `__init..`

Use `DEFINE_STATIC_CALL...`

Slots for call instructions at compile time.



`bprm_check_security` call slots patched at `__init`


SELINUX_SLOT
call selinux_bprm_check_security
AA_SLOT
call aa_bprm_check_security
BPF_SLOT
nop
Patched only when there is a BPF program attached.

Progress on `DEFINE_STATIC_CALL` is slow...

Upcoming..

BPF Ring Buffer

Merged!

Storage blobs a.k.a `bpf_local_storage`

patches on the list!

Sleepable BPF

patches on the list!

`bpf_d_path` helper

patches on the list!

Advanced string helpers (argv, file paths..)

Not started
(Custom Patches)

Load BPF programs during boot..

Not started...

Thank You!