<!-- Use http://vger.kernel.org/~acme/bpf/eBPF-lockdown-lpc-2020/?print-pdf#/ then control+P to print to pdf file -->

BPF LOCKDOWN

Using eBPF in kernel lockdown mode

Arnaldo Carvalho de Melo
acme@kernel.org
Red Hat Inc.

WHAT IS THIS ABOUT?

- kernel lockdown
- Cryptographic signature of eBPF bytecode
- Limiting access to confidential information
- libbpf code patching
- Problem statement
- No code written so far

KERNEL LOCKDOWN MODE

- Barrier between root and the kernel
- Integrity mode
- Confidentiality mode
- Initial merge: v5.4 (August 2019)

INTEGRITY MODE

- kernel and modules signed

INTEGRITY MODE

- kernel and modules signed
- eBPF bytecode signed

INTEGRITY MODE

- kernel and modules signed
- eBPF bytecode signed
- XDP offload

INTEGRITY MODE

- kernel and modules signed
- eBPF bytecode signed
- XDP offload
- Host checks signature

INTEGRITY MODE

- kernel and modules signed
- eBPF bytecode signed
- XDP offload
- Host checks signature
- Was it tested by a trusted party?

CONFIDENTIALITY MODE

- Integrity mode

CONFIDENTIALITY MODE

- Integrity mode
- Plus restrictions to accessing memory

EXAMPLES

- bpf_probe_read_kernel() and bpf_read_kernel_str()
- Should be already in compliance
- Restrictions to accessing memory

EXAMPLES

- bpf_probe_read_kernel() and bpf_read_kernel_str()
- Should be already in compliance
- Restrictions to accessing memory
- But not bpf_probe_read_kernel_user()?

EXAMPLES

- bpf_probe_read_kernel() and bpf_read_kernel_str()
- Should be already in compliance
- Restrictions to accessing memory
- But not bpf_probe_read_kernel_user()?

```
1  static __always_inline int
2  bpf_probe_read_kernel_common(void *dst, u32 size, const void *unsafe_ptr)
3  {
4          int ret = security_locked_down(LOCKDOWN_BPF_READ);
5          if (unlikely(ret < 0))
6                  goto fail;
7          ret = copy_from_kernel_nofault(dst, unsafe_ptr, size);
8          if (unlikely(ret < 0))
9                  goto fail;
10         return ret;
11 fail:   memset(dst, 0, size);
12         return ret;
13 }
```

EXAMPLES

- bpf_probe_read_kernel() and bpf_read_kernel_str()
- Should be already in compliance
- Restrictions to accessing memory
- But not bpf_probe_read_kernel_user()?

```
1  static __always_inline int
2  bpf_probe_read_kernel_common(void *dst, u32 size, const void *unsafe_ptr)
3  {
4          int ret = security_locked_down(LOCKDOWN_BPF_READ);
5          if (unlikely(ret < 0))
6                  goto fail;
7          ret = copy_from_kernel_nofault(dst, unsafe_ptr, size);
8          if (unlikely(ret < 0))
9                  goto fail;
10         return ret;
11 fail:   memset(dst, 0, size);
12         return ret;
13 }
```

SELINUX LOCKDOWN

Sample AVC audit output from denials:

```
avc:  denied  { integrity } for pid=3402 comm="fwupd"
 lockdown_reason="/dev/mem,kmem,port" scontext=system_u:system_r:fwupd_t:s0
 tcontext=system_u:system_r:fwupd_t:s0 tclass=lockdown permissive=0

avc:  denied  { confidentiality } for pid=4628 comm="cp"
 lockdown_reason="/proc/kcore access"
 scontext=unconfined_u:unconfined_r:test_lockdown_integrity_t:s0-s0:c0.c1023
 tcontext=unconfined_u:unconfined_r:test_lockdown_integrity_t:s0-s0:c0.c1023
 tclass=lockdown permissive=0
```

LOCKDOWN CHECK POINTS

```
$ find . -name "*.c" | xargs grep security_locked_down  | wc -l
59
$ find . -name "*.c" | xargs grep security_locked_down  | head
./arch/x86/kernel/ioport.c:                     security_locked_down(LOCKDOWN_IOPORT)))
./arch/x86/kernel/ioport.c:                security_locked_down(LOCKDOWN_IOPORT))
./arch/x86/kernel/msr.c:        err = security_locked_down(LOCKDOWN_MSR);
./arch/x86/kernel/msr.c:                err = security_locked_down(LOCKDOWN_MSR);
./arch/x86/mm/testmmiotrace.c:  int ret = security_locked_down(LOCKDOWN_MMIOTRACE);
./arch/powerpc/xmon/xmon.c:             lockdown = !!security_locked_down(LOCKDOWN_XMON_RW);
./arch/powerpc/xmon/xmon.c:             xmon_is_ro = !!security_locked_down(LOCKDOWN_XMON_WR);
./fs/proc/kcore.c:      int ret = security_locked_down(LOCKDOWN_KCORE);
./fs/debugfs/file.c:    if (security_locked_down(LOCKDOWN_DEBUGFS))
./fs/debugfs/inode.c:   int ret = security_locked_down(LOCKDOWN_DEBUGFS);
$
```

LOCKDOWN REASONS

```
enum lockdown_reason {
        LOCKDOWN_NONE,                  LOCKDOWN_MODULE_SIGNATURE,
        LOCKDOWN_DEV_MEM,               LOCKDOWN_EFI_TEST,
        LOCKDOWN_KEXEC,                 LOCKDOWN_HIBERNATION,
        LOCKDOWN_PCI_ACCESS,            LOCKDOWN_IOPORT,
        LOCKDOWN_MSR,                   LOCKDOWN_ACPI_TABLES,
        LOCKDOWN_PCMCIA_CIS,            LOCKDOWN_TIOCSSERIAL,
        LOCKDOWN_MODULE_PARAMETERS,     LOCKDOWN_MMIOTRACE,
        LOCKDOWN_DEBUGFS,               LOCKDOWN_XMON_WR,
        LOCKDOWN_INTEGRITY_MAX,
        LOCKDOWN_KCORE,                 LOCKDOWN_KPROBES,
        LOCKDOWN_BPF_READ,              LOCKDOWN_PERF,
        LOCKDOWN_TRACEFS,               LOCKDOWN_XMON_RW,
        LOCKDOWN_CONFIDENTIALITY_MAX,
};
```

WHAT OTHER BPF HELPERS TO LIMIT?

- When in doubt: restrict
- Open as we understand use case

SIGNING BPF

- Initially for pre-built bytecode

SIGNING BPF

- Initially for pre-built bytecode
- Like tools/bpf/runqslower/

SIGNING BPF

- Initially for pre-built bytecode
- Like tools/bpf/runqslower/
- bpftrace-like, dynamic, later

SIGNING BPF

- Initially for pre-built bytecode
- Like tools/bpf/runqslower/
- bpftrace-like, dynamic, later
- Reuse module signing utility

SIGNING BPF

- Initially for pre-built bytecode
- Like tools/bpf/runqslower/
- bpftrace-like, dynamic, later
- Reuse module signing utility
- Add signature to bpf_attr

SIGNING BPF

- Initially for pre-built bytecode
- Like tools/bpf/runqslower/
- bpftrace-like, dynamic, later
- Reuse module signing utility
- Add signature to bpf_attr
- Reuse module verification in kernel

BPFTOOL

- New 'sign' command
- Sign the ELF file

BPFTOOL

- New 'sign' command
- Sign the ELF file
- Sign each ELF section

BPFTOOL

- New 'sign' command
- Sign the ELF file
- Sign each ELF section
- Reuse scripts/sign-file.c

scripts/sign-file.c

```
$ ../build/v5.9-rc2+/scripts/sign-file
Usage: scripts/sign-file [-dp] <hash algo> <key> <x509> <module> [<dest>]
       scripts/sign-file -s <raw sig> <hash algo> <x509> <module> [<dest>]
$
```

LIBBPF

- Notices signature
- Adds it to the PROG_LOAD bpf_attr

KERNEL

- Notices signature
- Checks it like with kernel modules
- Hopefully shares the same signature verifier

module_sig_check()

```
static int module_sig_check(struct load_info *info, int flags)
{
        int err = info->len ? mod_verify_sig(info->hdr, info) : -ENODATA;
        switch (err) {
        case 0: info->sig_ok = true; return 0;
        case -ENODATA: reason = "Loading of unsigned module";                goto decide;
        case -ENOPKG:  reason = "Loading of module with unsupported crypto"; goto decide;
        case -ENOKEY:  reason = "Loading of module with unavailable key";
        decide:
                if (is_module_sig_enforced()) {
                        pr_notice("%s: %s is rejected\n", info->name, reason);
                        return -EKEYREJECTED;
                }
                return security_locked_down(LOCKDOWN_MODULE_SIGNATURE);
```

NO PROBLEMS?

- No code patching

NO PROBLEMS?

- No code patching
- CO-RE not involved

NO PROBLEMS?

- No code patching
- CO-RE not involved
- Signature verified

NO PROBLEMS?

- No code patching
- CO-RE not involved
- Signature verified
- Bytecode proceeds to the eBPF verifier

NO PROBLEMS?

- No code patching
- CO-RE not involved
- Signature verified
- Bytecode proceeds to the eBPF verifier
- End of story

BZZT: NEW CLASS OF FAILURE!

- Tainted signature

BZZT: NEW CLASS OF FAILURE!

- Tainted signature
- Code patching, CO-RE

BZZT: NEW CLASS OF FAILURE!

- Tainted signature
- Code patching, CO-RE
- BTF adjustments to struct fields

BZZT: NEW CLASS OF FAILURE!

- Tainted signature
- Code patching, CO-RE
- BTF adjustments to struct fields
- enumerator fixups

BZZT: NEW CLASS OF FAILURE!

- Tainted signature
- Code patching, CO-RE
- BTF adjustments to struct fields
- enumerator fixups
- Fallback to bpf_probe_read

BZZT: NEW CLASS OF FAILURE!

- Tainted signature
- Code patching, CO-RE
- BTF adjustments to struct fields
- enumerator fixups
- Fallback to bpf_probe_read
- bpf-to-bpf calls, dead code elimination

BZZT: NEW CLASS OF FAILURE!

- Tainted signature
- Code patching, CO-RE
- BTF adjustments to struct fields
- enumerator fixups
- Fallback to bpf_probe_read
- bpf-to-bpf calls, dead code elimination
- Others, more to come

WORKING AROUND

- Code patching highlighted

WORKING AROUND

- Code patching highlighted
- libbpf logs changes made

WORKING AROUND

- Code patching highlighted
- libbpf logs changes made
- Rebuild + re-sign for that kernel?

WORKING AROUND

- Code patching highlighted
- libbpf logs changes made
- Rebuild + re-sign for that kernel?
- To avoid CO-RE?

NOT POSSIBLE?

- Move parts of libbpf to kernel

NOT POSSIBLE?

- Move parts of libbpf to kernel
- Code patching

NOT POSSIBLE?

- Move parts of libbpf to kernel
- Code patching
- After signature verification

NOT POSSIBLE?

- Move parts of libbpf to kernel
- Code patching
- After signature verification
- User mode helper/driver?

USER MODE HELPER/DRIVER

- Like with bpfilter

USER MODE HELPER/DRIVER

- Like with bpfilter
- Using a binary format

USER MODE HELPER/DRIVER

- Like with bpfilter
- Using a binary format
- bpf.o executable

USER MODE HELPER/DRIVER

- Like with bpfilter
- Using a binary format
- bpf.o executable
- Running it triggers bytecode load

USER MODE HELPER/DRIVER

- Like with bpfilter
- Using a binary format
- bpf.o executable
- Running it triggers bytecode load
- Signature verification

USER MODE HELPER/DRIVER

- Like with bpfilter
- Using a binary format
- bpf.o executable
- Running it triggers bytecode load
- Signature verification
- Code patching

USER MODE HELPER/DRIVER

- Like with bpfilter
- Using a binary format
- bpf.o executable
- Running it triggers bytecode load
- Signature verification
- Code patching
- Hand it off to the verifier

QUESTIONS?

QUESTIONS?

SUGGESTIONS?

QUESTIONS?

SUGGESTIONS?

SOLUTIONS?

THE END

- "eBPF in Lockdown mode (reveal.js)"
- "Why lock down the kernel?", Matthew Garrett
- "Linux kernel lockdown, integrity, and confidentiality", Matthew Garrett
- "Restrict bpf when kernel is in lockdown confidentiality mode", David Howells
- "Kernel module signing facility", The Linux kernel user’s and administrator’s guide
- "BPF CO-RE (Compile Once – Run Everywhere)", Andrii Nakryiko
- "Rethinking bpfilter and user-mode helpers", Jonathan Corbet