Android Kernel Security

Linux Security Summit
Aug 2018

Jeff Vander Stoep and Sami Tolvanen
Acknowledgements

People who have reported security vulnerabilities to Android security:
https://source.android.com/security/overview/acknowledgements

Android SDL team
Evgenii Stepanov
Ivan Lozano
Joel Galenson
Vishwath Mohan
This data is public

https://source.android.com/security/bulletin/

Data: Sep 2017 → May 2018
Android is an open source project

Patches accepted!
Kernel accounts for $\frac{1}{3}$ of security vulnerabilities on Android.

Data: Sep 2017 → May 2018 (Android Oreo)
What’s working well: Attack Surface Reduction
“We think that by far the most effective mitigation work that we’ve seen on the Android platform over the last three years has been the investment in attack surface reduction. The deployment and tightening of selinux policies and the addition of seccomp sandboxing both result in an attacker needing to find more vulnerabilities in a smaller attack surface.”

Mark Brand - Google Project Zero
Access controls are “hard” mitigations which can be applied without knowledge of exploitation techniques.
Kernel vulnerabilities that are reachable in userspace but unreachable by unprivileged processes. (su → kernel vulns are excluded)

Data: Sep 2017 → May 2018 (Android Oreo)
Access control mechanism

- **SELinux**
  - e.g. CVE-2018-5858

- **Unix Permissions**
  - e.g. CVE-2017-14892

- **Capabilities**
  - e.g. CVE-2017-17712

(userspace reachable)

Sep 2017 → Apr 2018
Starting in Android Oreo all apps run with a seccomp filter.

e.g. Blocks CVE-2017-14140
Access control is effective

Attack surface reduction works!

Kernel provided access control + separation of privilege can substantially mitigate risks to the kernel.
Unprivileged reachable bugs

Some futex() and meltdown vulns.

(userspace reachable)
Sep 2017 → Apr 2018
Other userspace → kernel mitigations
Hardened Usercopy

Provides some run-time checks on data copied to/from userspace

copy_*_user()

Vulnerabilities by root cause (userspace reachable)

Sep 2017 → Apr 2018
Prevents direct kernel access to userspace.

Enforces use of (hardened) copy_*_user functions.

Found/fixed multiple instances of kernel directly accessing userspace.
Unfortunately, not all kernel vulns are reached via userspace.
Kernel vuln reachability

We’ve been discussing this.

But what about this ?!?!?

Data: Sep 2017 → May 2018
Non-userspace reachable vulns

By access vector
- wifi firmware
- wifi remote
- USB
- DSP
- modem firmware
- bluetooth remote
- modem
- bluetooth firmware

By root cause
- Missing Or Incorrect Bounds Check
- Integer Overflow
- Improper Input Validation
- Crypto Improperly Used
- Other

Data: Sep 2017 → May 2018
(a) The attack surface reduction tools provided by the kernel have been very effective on Android.

(b) In addition to attack surface reduction, the kernel now provides mechanisms such as hardened-usercopy + PAN which mitigate some userspace-reachable vulnerabilities.

(c) However, 1/3 of Android’s kernel bugs are reached by other vectors. We need tools similar to (a) and (b) to help address other access vectors.
Memory (un)safety

All kernel bugs
Data: May 2017 → May 2018
Control Flow Integrity
Control Flow Integrity

**What?**
Helps protect against code reuse attacks by adding runtime checks to ensure control flow stays within a precomputed graph.

**Where?**
LLVM $\geq$ 3.7 implements forward-edge CFI, which protects indirect branches.

**How?**
Allows an indirect branch only to the beginning of a valid function with the correct type.
How effective is CFI?

55% of indirect calls have ≤ 5 allowed targets

7% have > 100 allowed targets
LLVM’s CFI implementation requires LTO to determine all valid call targets.

Must use LLVM’s integrated assembler for inline assembly and an LTO-aware linker, i.e. GNU gold or LLVM lld.

Nearly all problems caused by toolchain compatibility issues. No kernel stability issues during several months of testing.
Link Time Optimization

Front-end

- .c
- .c
- .S

- .bc
- .bc
- .o

Thin archive

Linker

- Combined bitcode
- Optimizer
- Code generation

vmlinux
First Android devices with LTO+CFI kernels ship later this year.
C compilers allow indirect calls with mismatching types. Several benign CFI failures that had to be fixed.

Cross-DSO CFI support needed for kernel modules.

CFI adds a small overhead to indirect calls. Thanks to LTO, overall performance improved despite CFI.
Example of a CFI failure

Mismatching function pointer type

LLVM limits indirect calls to functions that match the type of the function pointer.

drivers/media/v4l2-core/v4l2-ioctl.c:

```c
if (info->flags & INFO_FL_STD) {
    typedef int (*vidioc_op)(struct file *file, void *fh, void *p);
    const void *p = vfd->ioctl_ops;
    const vidioc_op *vidioc = p + info->u.offset;

    ret = (*vidioc)(file, fh, arg);
}
```

Fixed in 3ad3b7a2ebaefae3 ("media: v4l2-ioctl: replace IOCTL_INFO_STD with stub functions")
CFI check, slowpath for cross-DSO. Only returns if the target address is allowed.

Indirect function call.
Error handling

In normal mode, CFI failure results in a kernel panic, which includes the target address.

For debugging only, a permissive mode that produces a warning instead.

CFI failure (target: [ffffffff3e83d4d80]
my_target_function+0x0/0xd80):
------------[ cut here ]------------
kernel BUG at kernel/cfi.c:32!
Internal error: Oops - BUG: 0 [#1] PREEMPT SMP
...
Call trace:
...
[ffffffff8752d00084>] handle_cfi_failure+0x20/0x28
[ffffffff8752d00268>] my_buggy_function+0x0/0x10
...
CFI in Android kernels

Supported in 4.9 and 4.14 for arm64

CONFIG_LTO_CLANG=y
CONFIG_CFI_CLANG=y

# CONFIG_CFI_PERMISSIVE=y for debugging.

Requires clang ≥ 5.0 and binutils ≥ 2.27.
CFI only protects indirect branches. LLVM’s Shadow Call Stack helps protect return addresses.

Plenty of issues with GNU gold. Ongoing work to switch to LLVM’s lld linker instead.
Thank you