DEFInit: An Analysis of Exposed Android Init Routines

Yuede Ji*
University of North Texas

*This work was done while interning at Kryptowire.

Mohamed Elsabagh, Ryan Johnson, Angelos Stavrou
Kryptowire
Android Init

Init Process

- Common *NIX process
- First process in user-space
- Highly-privileged process

Unique in Android

- Acts as the system property store
- Supports Init Routines
Android Init Routines

Programs executed by Android Init in response to changes to system properties

- Customizable by vendors
- Only system (privileged; pre-installed) apps/processes can set system properties
Android Init Routines

Background:

System App

App

App

Android Framework

Zygote (ART VM)

Init Routines

Daemons

Init

Kernel

Bootloader

(highly privileged)

(highly privileged)

(lower privilege)
Background:

Android Init Routines
Background:

Android Init Routines
Background:

**Android Init Routines**

👍 Unique hardware features, diagnostics, docking, mounting, etc., that cannot be done directly in a privileged app
Background:

Android Init Routines

⚠️ Improper access control in privileged apps can expose Init Routines to unprivileged apps, resulting in crossing security boundaries!
What We Found

Numerous zero-day privilege-escalation vulnerabilities due to custom (added by vendors) Init routines that are exposed to unprivileged apps!
Real-World Example

Attack App (unprivileged)

void exploit() {
    v0 = new Intent("wifitest");
    sendBroadcast(v0);
    ...
}

System App (privileged)

void onReceive(...) {
    ...
    setprop("a", "1");
    setprop("b", "true");
    ...
}

Init Routines (.rc file)

on property:a=1 && property:b=true
    ...
    start wifitest
    ...

service wifitest /bin/wifitest.sh
    user root
    ...

/bin/wifetest.sh

    ...
    setenforce 0
    ...

Disable SELinux!
Automated Discovery with DEFInit

- **Automated system** to identify exposed security-sensitive Init routines, their behaviors, and the apps exposing them

- **First study** on the security impact of customized Android Init routines

- **89 High-Impact Zero-Days**
  - Disabling SELinux, sniffing network traffic, reading system logs, recording screen, etc.
Unpack and Extract Init Routine Definitions

Challenges:

- Multiple firmware file formats
- Dynamically load and process .rc files
- Import Init sections defined in other files
- Service and action definitions are polymorphic

Solutions:

- Integrate multiple firmware file format unpackers
- Start parsing at the root /init.rc file
- Nest into imported files in depth-first order
- Keep track of merge or override options
DEFInit:

Estimate Behaviors of Init Routines

Challenges:

- Estimating security-relevant behavior of arbitrary programs
- Multiple formats: Init commands, ELF binaries, Shell Scripts

Behavioral Rules

Collect Traces & Estimate Behaviors

r1: wifitest.sh: [setprop x y; setenforce 0; ...]
   (perms, disable selinux, ...)

r2: x.sh: [tcpdump a; mv a /sdcard/ ...],
   (dump, capture traffic, ...)

...
DEFInit:

Estimate Behaviors of Init Routines

Solutions:

- Extract code traces
  - ELF binaries → collect static traces of called APIs along CFG paths in DFS order
  - Shell scripts → collect code traces by dry-running them in a custom shell tracer

```
A = "//data/tmp.log"
logcat * > A
print(C)
mv(A, B)
B = "//sdcard"
C = "test"
```
DEFInit:

Estimate Behaviors of Init Routines

Solutions:

- Extract code traces
  - ELF binaries → collect static traces of called APIs along CFG paths in DFS order
  - Shell scripts → collect code traces by dry-running them in a custom shell tracer
- Match interesting trace sequences using static rules

Read Logcat Logs:

```
logcat * (-f>*) $sdcard/*
| logcat * (-f>*) *
(mv|cp) * $sdcard/*
```
DEFInit:

Model Trigger Conditions

Challenges

- Multiple interdependencies between Init actions, services, Android commands, and APIs
DEFinit: Model Trigger Conditions

Challenges

- Multiple interdependencies between Init actions, services, Android commands, and APIs

Solutions

- A novel graph structure called an Init Dependency Graph (IDG)
DEFInit:

Identify Exposed Routines and Behaviors

Challenges

- Identifying system property call sites
- Resolving argument values
- Detecting if a call site is exposed to unprivileged apps

Solutions

- Context- and flow-sensitive property key and value extraction
- Reachability query from IPC entry points to call sites on control-flow graphs
Evaluation:

Analyzed Firmware Dataset

- 259 Android 8 -- 11 firmware
- 21 top vendors worldwide
- 65k system apps
  - 262 per firmware (average)
Evaluation:

Identified Init Routines

- 223 routines per firmware
- 66% custom (added by vendors)

Exposed Init Routines Per Firmware

- 8 exposed routines per firmware
- 2 sensitive exposed routines per firmware
- \textit{All} custom!
Evaluation:

Exposed Behaviors

- 515 exposed sensitive functionalities
  - 430 unique routines
  - 173 unique apps
  - 101 firmware
  - 13 vendors
Evaluation:

**Discovered Zero-Day Vulnerabilities**

- Verified 89 vulnerabilities
  - 34 unique apps
  - 30 firmware
  - 6 vendors
- Developed+Tested 59 PoCs
- 49 confirmed by 3 vendors so far
Evaluation:

Impact of Discovered Vulnerabilities

- Disable system-wide Mandatory Access Control
- Execute arbitrary code with high privileges
- Spy on screen content and user activity
- Spy on content of sent/received SMS messages and Calls
- Capture data transmitted using Wi-Fi and Bluetooth
- Capture an extensive amount of PII from numerous sources
- Capture information to attack other processes (mmap, open files, etc.)
- Prevent meaningful usage of the device (persistent DoS attack)
- Access stored Wi-Fi passwords
- ...

Who hacked my Android?

Courtesy: thehackernews.com
DEFInit: An Analysis of Exposed Android Init Routines

- Systematically studied security impact of Android Init routines
- Novel study and automated analysis
- Various high-impact zero-days in Android 8 -- 11 devices
- More in paper
  - Routines exposed via the GUI, rule samples, characteristics of exposed routines, commands called by exposed routines, etc.

Thank You!

yuede.ji@unt.edu, melsabagh@kryptowire.com, rjohnson@kryptowire.com, astavrou@kryptowire.com