Come Join the [CAFSA] - Continuous Automated Firmware Security Analysis

Collin Mulliner

cruise

Black Hat USA, August 2019

Collin Mulliner

Building Embedded Devices on and off since 2006

Developing Mobile Apps since 1997

(In-)Security since ‘90s

Research/Academia with focus on Mobile Device & App Security ~10 years

Currently: Principal Security Engineer, Autonomous Vehicle Security @ Cruise

@collinrm

Building Secure Devices is Hard

Devices

Industrial & IOT

source: https://developer.qualcomm.com/hardware/dragonboard-410c

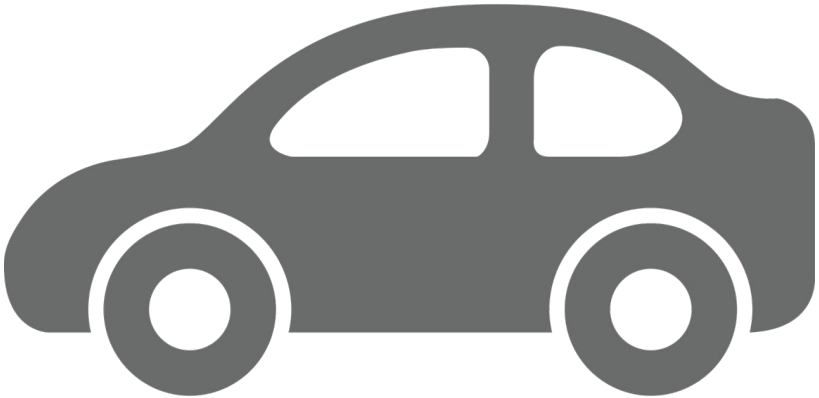
Customized Android Devices

source: https://www.amazon.com/Huawei-MediaPad-Kardon-Tuned-Speakers-Warranty/dp/B07M5NLGVY

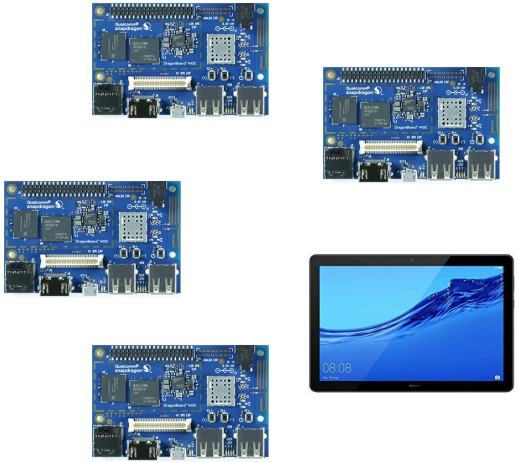
Consumer Electronics

source: https://zionmarketresearch.wordpress.com/2017/05/22/consumer-electronics-market/

Car → Devices on Wheels



=



Devices & Firmware

Device with full operating system: kernel and user-space, actual filesystem

- Firmware consists of kernel and a number filesystem images

Device without kernel, no kernel/user-space just a number of tasks

- Firmware consist of one binary that has code and data

Devices & Firmware

Device with full operating system: kernel and user-space, actual filesystem

- **Firmware consists of kernel and a number of filesystem images**

Device without kernel, no kernel/user-space just a number of tasks

- Firmware consist of one binary that has code and data

Basically this covers anything that runs Linux (incl. Android) and similar (e.g. QNX)


FIRMWARE
LIGHTYEAR
FIRMWARE, EVERYWHERE
imgflip.com

Firmware = Your Product = Firmware

Firmware Security

Security Features:

- Secure Boot : valid signatures for kernel and filesystems
- Configuration : production version of configuration files
- Binaries : only production binaries → no debug/dev/test binaries
- Access Control : file permissions and labels
- Keys/Certs : any kind of cryptographic material
- Hardened Code : production build, hardening options, no symbols, ...

Firmware is “secure” if all security features are combined in production

Firmware Bugs

CVE-2017-8222	Wireless IP Camera (P2P) WIFICAM devices have an "Apple Production IOS Push Services" private RSA key and certificate stored in /system/www/pem/ck.pem inside the firmware, which allows attackers to obtain sensitive information.
CVE-2014-5457	QNAP TS-469U with firmware 4.0.7 Build 20140410, TS-459U, TS-EC1679U-RP, and SS-839 use world-readable permissions for /etc/config/shadow, which allows local users to obtain usernames and hashed passwords by reading the password.
CVE-2018-19071	An issue was discovered on Foscam C2 devices with System Firmware 1.11.1.8 and Application Firmware 2.72.1.32, and Opticam i5 devices with System Firmware 1.5.2.11 and Application Firmware 2.21.1.128. /mnt/mtd/boot.sh has 0777 permissions, allowing local users to control the commands executed at system start-up.
CVE-2018-19072	An issue was discovered on Foscam C2 devices with System Firmware 1.11.1.8 and Application Firmware 2.72.1.32, and Opticam i5 devices with System Firmware 1.5.2.11 and Application Firmware 2.21.1.128. /mnt/mtd/app has 0777 permissions, allowing local users to replace an archive file (within that directory) to control what is extracted to RAM at boot time.
CVE-2017-14428	D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02) devices have 0666 /var/run/hostapd* permissions.
CVE-2017-14427	D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02) devices have 0666 /var/run/storage_account_root permissions.
CVE-2017-14426	D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02) devices have 0644 /var/etc/shadow (aka the /etc/shadow symlink target) permissions.
CVE-2017-14425	D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02) devices have 0666 /var/etc/hnapasswd permissions.
CVE-2017-14424	D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02) devices have 0666 /var/passwd permissions.
CVE-2018-20052	An issue was discovered on Cerner Connectivity Engine (CCE) 4 devices. The user running the main CCE firmware has NOPASSWD sudo privileges to several utilities that could be used to escalate privileges to root. One example is the "sudo ln -s /tmp/script /etc/cron.hourly/script" command.
CVE-2018-5399	The Auto-Maskin DCU 210E firmware contains an undocumented Dropbear SSH server, v2015.55, configured to listen on Port 22 while the DCU is running. The Dropbear server is configured with a hard-coded user name and password combination of root / amroot. The server is configured to use password only authentication not cryptographic keys, however the firmware image contains an RSA host-key for the server. An attacker can exploit this vulnerability to gain root access to the Angstrom Linux operating system and modify any binaries or configuration files in the firmware. Affected releases are Auto-Maskin DCU-210E RP-210E: Versions prior to 3.7 on ARMv7.

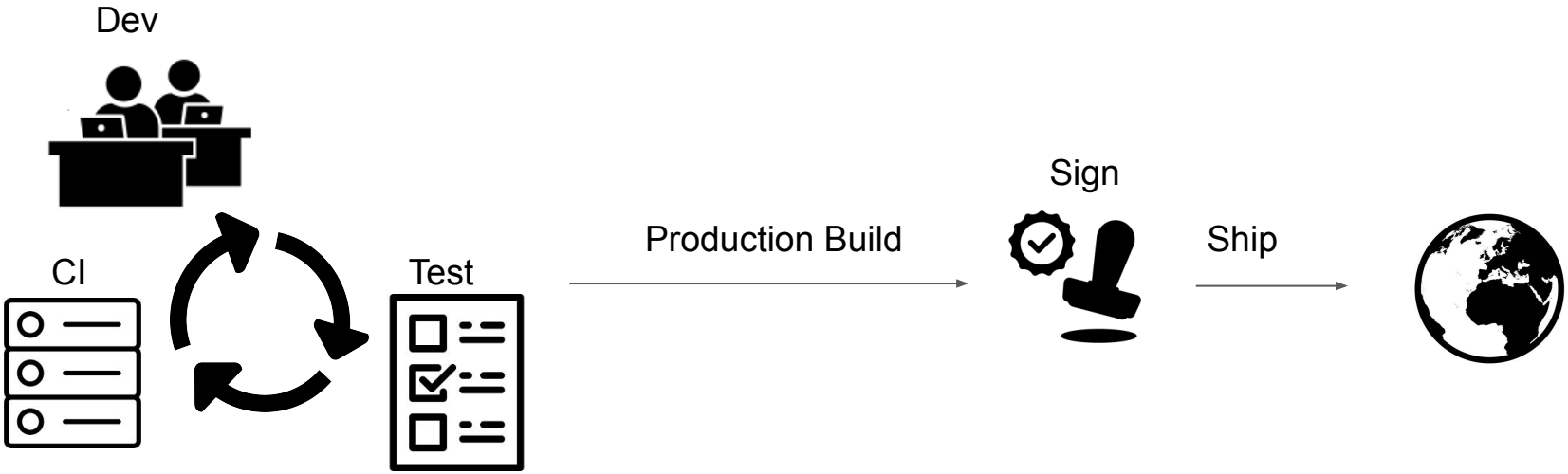
Firmware Bugs

CVE-2017-8222	Wireless IP Camera (P2P) WIFICAM devices have an "Apple Production IOS Push Services" private RSA key and certificate stored in /system/www/pem/ck.pem inside the firmware, which allows attackers to obtain sensitive information.
CVE-2014-5457	QNAP TS-469U with firmware 4.0.7 Build 20140410, TS-459U, TS-EC1679U-RP, and SS-839 use world-readable permissions for /etc/config/shadow, which allows local users to obtain usernames and hashed passwords by reading the password.
CVE-2018-19071	An issue was discovered on Foscam C2 devices with System Firmware 1.11.1.8 and Application Firmware 2.72.1.32, and Opticam i5 devices with System Firmware 1.5.2.11 and Application Firmware 2.21.1.128. /mnt/mtd/boot.sh has 0777 permissions, allowing local users to control the commands executed at system start-up.
CVE-2018-19072	An issue was discovered on Foscam C2 devices with System Firmware 1.11.1.8 and Application Firmware 2.72.1.32, and Opticam i5 devices with System Firmware 1.5.2.11 and Application Firmware 2.21.1.128. /mnt/mtd/app has 0777 permissions, allowing local users to replace an archive file (within that directory) to control what is extracted to RAM at boot time.
CVE-2017-14428	D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02) devices have 0666 /var/run/hostapd* permissions.
CVE-2017-14427	D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02) devices have 0666 /var/run/storage_account_root permissions.
CVE-2017-14426	D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02) devices have 0644 /var/etc/shadow (aka the /etc/shadow symlink target) permissions.
CVE-2017-14425	D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02) devices have 0666 /var/etc/hnapasswd permissions.
CVE-2017-14424	D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02) devices have 0666 /var/passwd permissions.
CVE-2018-20052	An issue was discovered on Cerner Connectivity Engine (CCE) 4 devices. The user running the main CCE firmware has NOPASSWD sudo privileges to several utilities that could be used to escalate privileges to root. One example is the "sudo ln -s /tmp/script /etc/cron.hourly/script" command.
CVE-2018-5399	The Auto-Maskin DCU 210E firmware contains an undocumented Dropbear SSH server, v2015.55, configured to listen on Port 22 while the DCU is running. The Dropbear server is configured with a hard-coded user name and password combination of root / amroot. The server is configured to use password only authentication not cryptographic keys, however the firmware image contains an RSA host-key for the server. An attacker can exploit this vulnerability to gain root access to the Angstrom Linux operating system and modify any binaries or configuration files in the firmware. Affected releases are Auto-Maskin DCU-210E RP-210E: Versions prior to 3.7 on ARMv7.

Firmware Bugs, How?

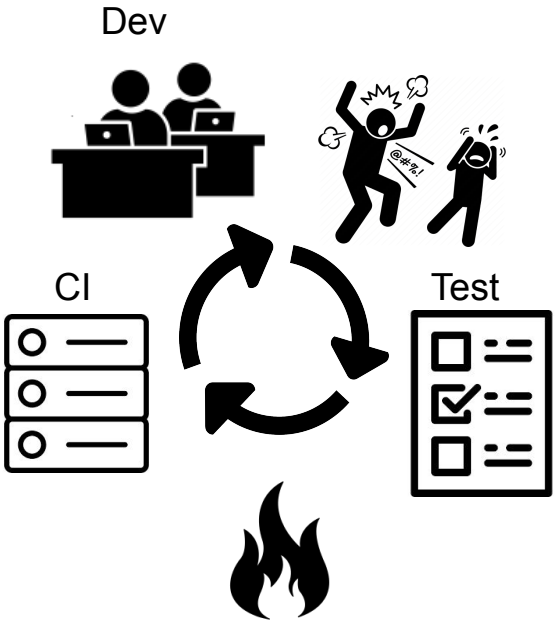
Firmware Changes Over Time

Prototyping, Development, Testing, QA, Pentest, …... Production



Firmware Changes Over Time - Reality

Prototyping, Development, Testing, QA, Pentest, …... Production



- New feature
- Disabled security setting to debug
- New feature
- Need to open that network port
- Developers like to ssh into the device
- New feature
- QA needs to do X, make special build
- Removed functionality Z
- Y only works on dev, lets change prod

Lots and Lots of Changes

The Ripple Effect of shipping ‘bad’ Firmware

Non or partially working Security Controls

(Partially) defective Product

Firmware update mechanism crippled

Long term Negative Impact on: Product, Company, and Ecosystem

- Loss of Reputation
- High Costs in case of a Recall

Automating Firmware Security QA

1. **Tool to perform automated firmware security checks**

2. **Deploy tool to strategic locations in your development and release flow**

3. **Feed results back to developers (and suppliers)**

FwAnalyzer : a Tool for Firmware Security QA

Expert creates Rules → FwAnalyzer enforces Rules and produces a Report

Rules are applied to Files within FileSystem images of Firmware

- Enforce : type, ownership, permissions, security labels
- Analyze File Content : compare digest, apply RegEx, run external program

FwAnalyzer : a Tool for Firmware Security QA

Expert creates Rules → FwAnalyzer enforces Rules and produces a Report

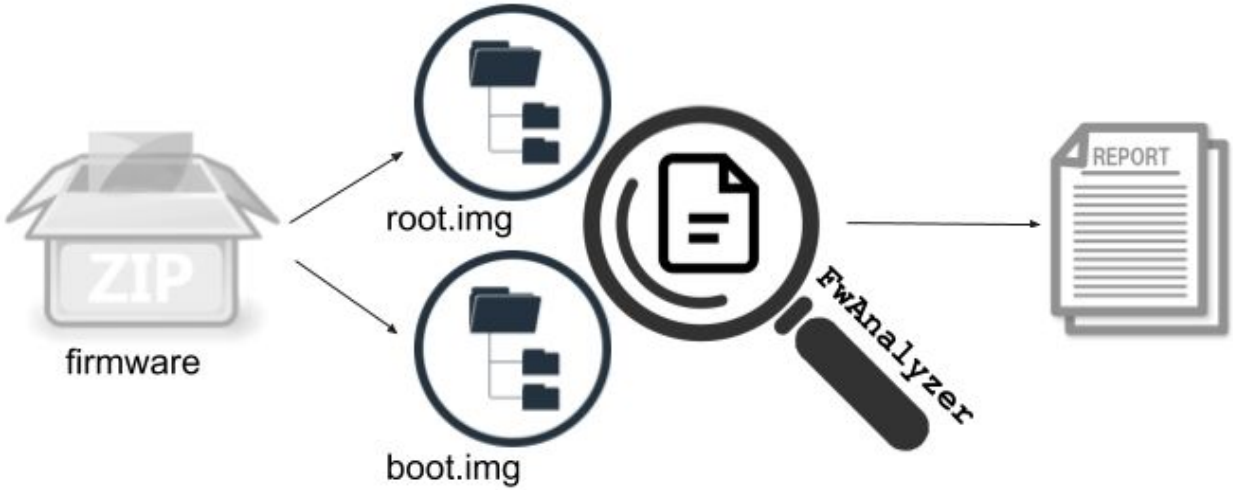
Rules are applied to Files within FileSystem images of Firmware

- Enforce : type, ownership, permissions, security labels
- Analyze File Content : compare digest, apply RegEx, run external program

Use Cases:

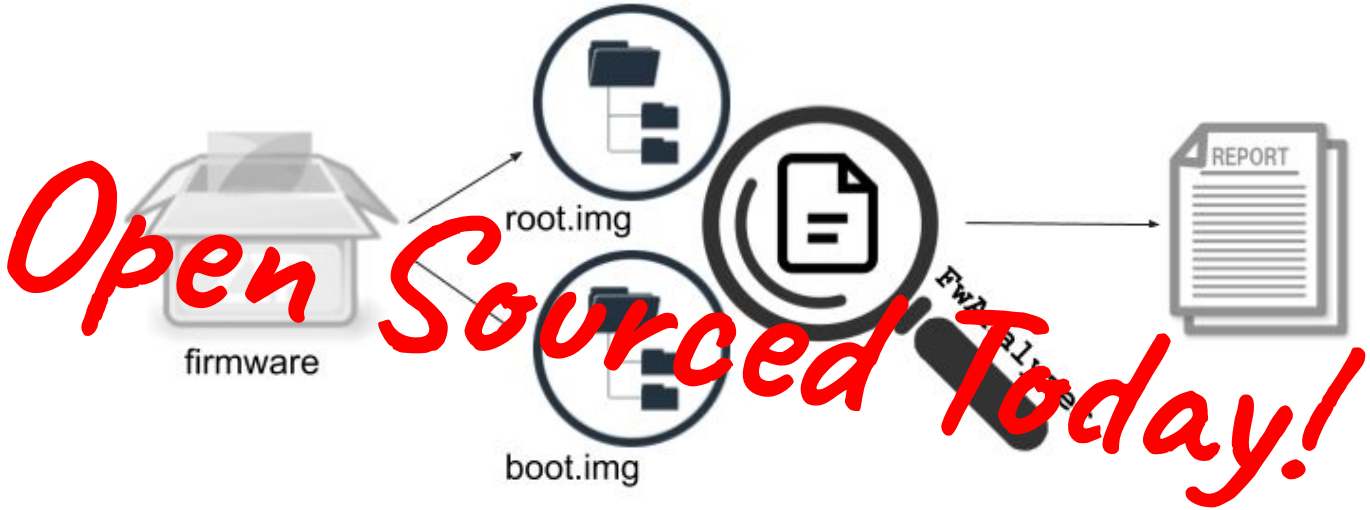
- Run in CI during development → produce immediate feedback
- Gate Production Signing → don't sign 'insecure' firmware with prod keys
- Acceptance Testing for 3rd party firmware → don't deploy bad firmware
- …..

FwAnalyzer : a Tool for Firmware Security QA



Source: http://www.github.com/cruise-automation/fwanalyzer

FwAnalyzer : a Tool for Firmware Security QA



Source: http://www.github.com/cruise-automation/fwanalyzer

Goal

Prevent ‘bad’ high impact changes from going into Production

- Debugging features or weak configuration
- Signing ‘bad’ builds with production keys

Prevent a ripple effect due to bad security!

You put a lot of effort into security features. Make sure they are enabled!

Create (self) confidence about the security of your product!

Additional Goals

Visibility: information that is hard to acquire but super helpful

- version information, security parameters, ...

Automation: designed for integration into development pipeline

- produce machine readable output for post processing

Collaboration: use rules from external sources

- share ‘common’ rules but keep sensitive rules private

NOT IN SCOPE: Software Vulnerabilities

FwAnalyzer IS NOT a

...vulnerability finding tool to find bugs in source or binaries

...CVE scanner to analyze your software stack for known vulnerabilities

Use existing tools and processes to check your software!

*FwAnalyzer-scripts can be used to call such tools from within FwAnalyzer

FwAnalyzer

```
$ fwanalyzer -cfg system.toml -in system.img -out system_out.json
```

FwAnalyzer

```
$ fwanalyzer -cfg system.toml -in system.img -out system_out.json
```

```
[GlobalConfig]
FsType = "extfs"
FsTypeOptions = "selinux"
DigestImage = true

[GlobalFileChecks]
Suid = true
SeLinuxLabel = true
WorldWrite = true
Uids = [0,1000,1003,2000]
Gids = [0,1000,1003,2000]
BadFiles = ["/xbin/tcpdump", "/xbin/su"]

[FileTreeCheck]
OldTreeFilePath = "system_filetree.json"
CheckPermsOwnerChange = true
CheckFileSize = false

[Include."build_infos.toml"]
```

Rules

FwAnalyzer

```
$ fwanalyzer -cfg system.toml -in system.img -out system_out.json
```

FileSystem Image

```
[GlobalConfig]
FsType = "extfs"
FsTypeOptions = "selinux"
DigestImage = true

[GlobalFileChecks]
Suid = true
SeLinuxLabel = true
WorldWrite = true
Uids = [0,1000,1003,2000]
Gids = [0,1000,1003,2000]
BadFiles = ["/xbin/tcpdump", "/xbin/su"]

[FileTreeCheck]
OldTreeFilePath = "system_filetree.json"
CheckPermsOwnerChange = true
CheckFileSize = false

[Include."build_infos.toml"]
```

Rules

FwAnalyzer

```
$ fwanalyzer -cfg system.toml -in system.img -out system_out.json
```

```
[GlobalConfig]
FsType = "extfs"
FsTypeOptions = "selinux"
DigestImage = true

[GlobalFileChecks]
Suid = true
SeLinuxLabel = true
WorldWrite = true
Uids = [0,1000,1003,2000]
Gids = [0,1000,1003,2000]
BadFiles = ["/xbin/tcpdump", "/xbin/su"]

[FileTreeCheck]
OldTreeFilePath = "system_filetree.json"
CheckPermsOwnerChange = true
CheckFileSize = false

[Include."build_infos.toml"]
```

Rules


```
{
    "FSType": "extfs",
    "ImageName": "/home/cmulliner/research/talk/unpacked/system.img",
    "ImageDigest": "7896ae177e5504ca6d9706f2f896da483e423d68cccc398948fb8fa3f8751ccb",
    "CurrentFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json.new",
    "OldFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json",
    "Data": {
        "build": "production",
        "version": "23"
    },
    "Informational": {
        "/bin/zcat": [
            "CheckFileTree: new file: 120755 0:2000 6 0 SELinux label: u:object_r:system_file:s0"
        ]
    },
    "Offenders": {
        "/xbin/su": [
            "File is SUID, not allowed",
            "File not allowed"
        ],
        "/xbin/tcpdump": [
            "File not allowed"
        ]
    }
}
```


Report

FwAnalyzer

```
$ fwanalyzer -cfg system.toml -in system.img -out system_out.json
```

```
[GlobalConfig]
FsType = "extfs"
FsTypeOptions = "selinux"
DigestImage = true

[GlobalFileChecks]
Suid = true
SeLinuxLabel = true
WorldWrite = true
Uids = [0,1000,1003,2000]
Gids = [0,1000,1003,2000]
BadFiles = ["/xbin/tcpdump", "/xbin/su"]

[FileTreeCheck]
OldTreeFilePath = "system_filetree.json"
CheckPermsOwnerChange = true
CheckFileSize = false

[Include."build_infos.toml"]
```

Rules

issue detected


```
{
    "FSType": "extfs",
    "ImageName": "/home/cmulliner/research/talk/unpacked/system.img",
    "ImageDigest": "7896ae177e5504ca6d9706f2f896da483e423d68cccc398948fb8fa3f8751ccb",
    "CurrentFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json.new",
    "OldFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json",
    "Data": {
        "build": "production",
        "version": "23"
    },
    "Informational": {
        "/bin/zcat": [
            "CheckFileTree: new file: 120755 0:2000 6 0 SELinux label: u:object_r:system_file:s0"
        ]
    },
    "Offenders": {
        "/xbin/su": [
            "File is SUID, not allowed",
            "File not allowed"
        ],
        "/xbin/tcpdump": [
            "File not allowed"
        ]
    }
}
```


Report

Firmware vs FileSystem Image

Firmware = kernel + filesystem images bundled in one file (zip, cpio, ...)

FwAnalyzer targets factory image and/or full firmware update

- Reason: Uncertainty about firmware content until it is fully assembled
  - What happens when is highly dependant on the build system
  - Every platform is different

Side effect: automatic support for firmware of Commercial Off-The-Shelf devices

FileSystem Image vs. File

FileSystem has a lot of data vs. File just contains bytes

- File metadata
  - Owner
  - Type & Permission
  - Attributes (e.g., SELinux label)
- Directory structure

Compare entire FileSystem Images against each other

- Find new/modified/removed files and directories

FwAnalyzer Infrastructure

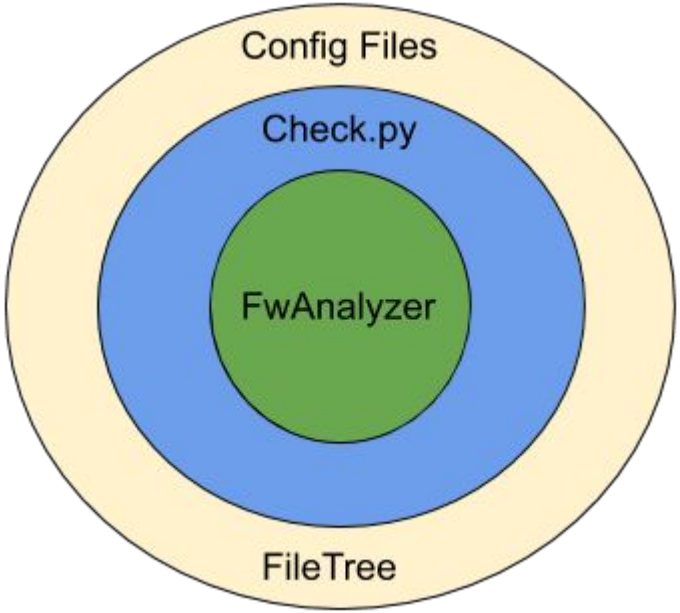
FwAnalyzer = the actual executable

`check.py`

- Unpack firmware and run FwAnalyzer

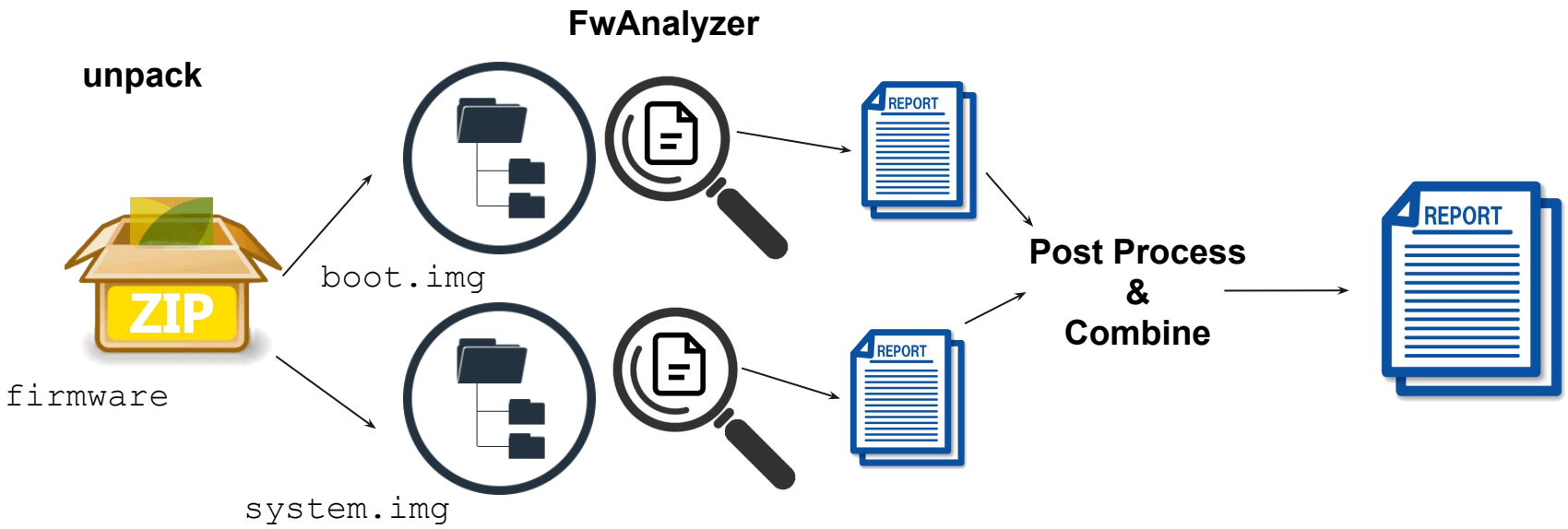
Config Files = Rules

- Configuration for each filesystem image
- Supports including files from “rule library”



FileTree = list of all files in filesystem image (from a previous run)

Workflow


FwAnalyzer
unpack
ZIP
firmware
boot.img
system.img
REPORT
REPORT
Post Process
&
Combine
REPORT

FwAnalyzer

Operates on the filesystem **image** and analyzes it without mounting it!

- Currently supports: ext2/4, vFat, squashfs, ubifs, and local directory

FwAnalyzer outputs JSON → easy post processing and automation

- offenders : violations of the configured rules → firmware fails the analysis
- informational : non fatal checks → visibility and rule testing
- data : extracted from filesystem → visibility and post processing

FwAnalyzer

Operates on the filesystem **image** and analyzes it without mounting it!

- Currently s ctory

FwAnalyzer out

- offenders ls the analysis
- information
- data rocessing


```
{
    "FSType": "extfs",
    "ImageName": "/home/cmulliner/research/talk/unpacked/system.img",
    "ImageDigest": "7896ae177e5504ca6d9706f2f896da483e423d68cccc398948fb8fa3f8751ccb",
    "CurrentFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json.new",
    "OldFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json",
    "Data": {
        "build": "production",
        "version": "23"
    },
    "Informational": {
        "/bin/zcat": [
            "CheckFileTree: new file: 120755 0:2000 6 0 SELinux label: u:object_r:system_file:s0"
        ]
    },
    "Offenders": {
        "/xbin/su": [
            "File is SUID, not allowed",
            "File not allowed"
        ],
        "/xbin/tcpdump": [
            "File not allowed"
        ]
    }
}
```

FwAnalyzer: Overview

Global Configuration

- FsType & Options

```
[GlobalConfig]
FsType = "extfs"
FsTypeOptions = ""
DigestImage = false
```

FileSystem “drivers” implemented via external tools

- e2tools, squashfs tools, ubireader, mtools, … (we patched some of these)

Checks implemented against `abstract file` object

- Checks don’t need to know about the filesystem type
- Extremely easy to add new checks

Global File Checks

Check every file in the filesystem, flagging offending files based on:

- SUID : flag SUID file (with possibility to whitelist files)
- WorldWrite : flag world writable file
- SELinuxLabel : flag UNLABELED file
- Allowed UID/GID : flag file not owned by allowed UID/GID
- BadFiles : flag file if on badfile list

These checks provide an easy starting point to define a general security model.

```
[GlobalFileChecks]
Suid = true
SuidWhiteList = ["/bin/runs-as"]
WorldWrite = true
Uids = [0,1000,1003,2000]
Gids = [0,1000,1003,2000]
BadFiles = ["/xbin/su", "/xbin/tcpdump"]
```

File Stat Check

Check the metadata of a specific file

- Mode : file type, permissions, flags
- Owner : UID and/or GID
- AllowEmpty : allow/disallow file size zero
- SELinuxLabel : the specific label
- LinkTarget : target of a softlink

```
[FileStatCheck."/etc/shadow"]
AllowEmpty = false
Uid = 0
Gid = 0
Mode = "100640"
```

Allows to model parameters of a specific file.

File Content

Check file content (using a number of different methods)

- RegEx : apply regular expression to the content of the file
- Digest : compare the digest to a configured digest
- JSON : parse file as JSON and compare a specific field to configured value
- Script : run an external program

Core functionality to enforce rules based on file content.

File Content: Examples

```
[FileContent."mount_flag_noexec"]
File = "/etc/fstab"
RegEx = ".*\\n/dev/sda1[\\t ]+/mnt[\\t ]+ext4[\\t a-z,]+noexec.*\\n.*"
Desc = "sda1 should be mounted noexec"
```

```
[FileContent."/etc/security/otacerts.zip"]
File = "/etc/security/otacerts.zip"
Digest = "d44ebdd80c1aa48565e626b9521b7bd484395a92865df293a3aac8f911f0bdab"
```

```
[FileContent."stripped binary"]
File = "/usr/bin/"
Script = "checked_stripped.sh"
Desc = "binary should be stripped"
```

```
[FileContent."dmverity"]
File = "/grub.cfg"
RegEx = ".*linux[[:alpha:]/ ]+root=/dev/dm-0[[:alnum:] ,=\"]+verity.*\\n.*"
Desc = "check dm-verity is enabled"
```

```
"/usr/bin/mydaemond": [
        "script(check_stripped.sh) returned=mydaemond is not stripped"
    ],
```

Extract Data

Extract file content (using a number of different methods)

- RegEx : regular expression with capture group
- JSON : parse file as JSON and extract specific field
- Script : stdout of external program

```
[DataExtract."Version"]
File = "/etc/release_ver"
RegEx = "^version=(\\S+)\\n.*"
```

→

```
"Data": {
        "Version": "1.33.7",
},
```

Core functionality to provide visibility

FileTree Check

FileTree → list of files in the filesystem (including metadata and file digest)

The filetree is collected every run and compared to the filetree of a previous run

FileTree Check

FileTree → list of files in the filesystem (including metadata and file digest)

The filetree is collected every run and compared to the filetree of a previous run


```
{
    "image_name": "test/test.img",
    "image_digest": "9d5fd9acc98421b46976f283175cc438cf549bb0607a1bca6e881d3e7f323794",
    "files": [
        {
            "size": 1024,
            "mode": 16877,
            "uid": 0,
            "gid": 0,
            "se_linux_label": "-",
            "name": "/dir1/dir11",
            "link_target": "",
            "digest": "0"
        },
        {
            "size": 0,
            "mode": 33188,
            "uid": 0,
            "gid": 0,
            "se_linux_label": "-",
            "name": "/dir1/file11",
            "link_target": "",
            "digest": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
        },
```

FileTree Check

FileTree → list of files in the filesystem (including metadata and file digest)

The filetree is collected every run and compared to the filetree of a previous run

Output: NEW files, DELETED files, and MODIFIED files

Provide an easy way to identify changes between two filesystems.

```
"Informational": {
  "/bin/zcat": [
    "CheckFileTree: new file: 120755 0:2000 6 0 SELinux label: u:object_r:system_file:s0"
  ]
},
```

Script - Implement Custom Checks

FwAnalyzer extracts File from FileSystem image to temp directory

Script run on extracted file and metadata is passed to script via cmdline args

Examples:

- Binaries stripped, DEP, ASLR, etc... enabled?
- Does file contain private key?
- Analyze application packages (e.g., APKs)
- Custom check
  - e.g., check that binary contains specific built-in config file

Checks and InformationalOnly

Checks produce **offenders**

- e.g. file has bad permission

Presence of **offenders** indicate security issues → bad firmware

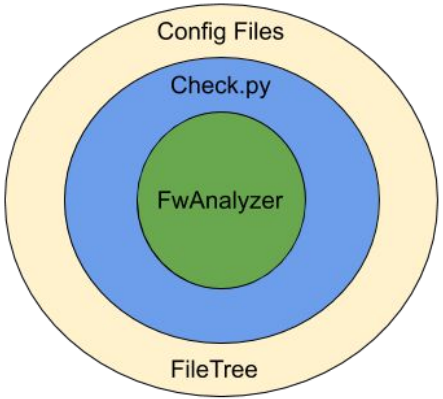
Informational results basically are ‘for your information’

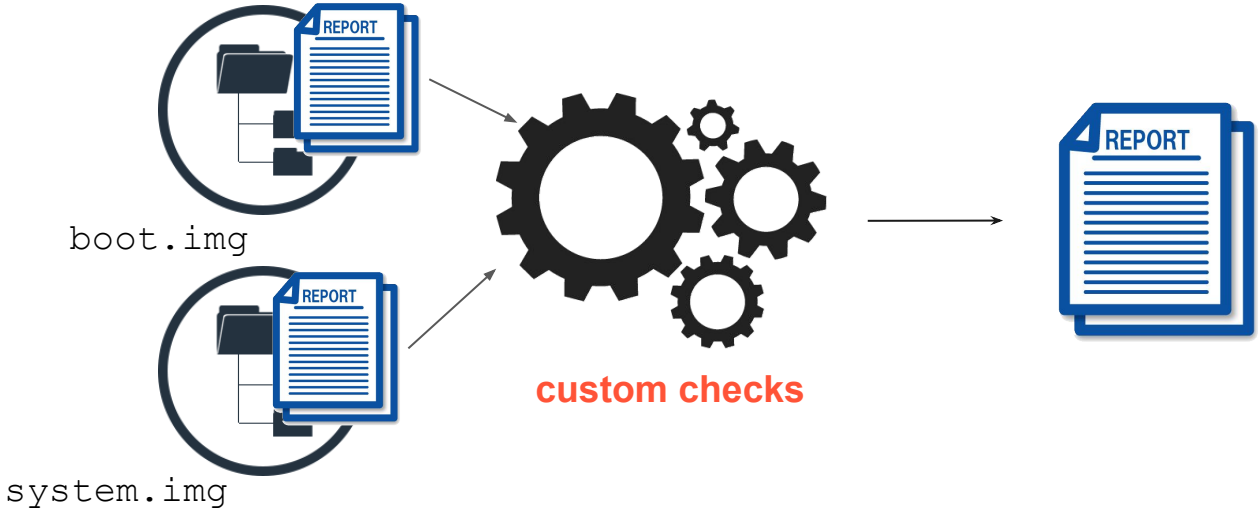
Set checks to **InformationalOnly** to *convert offender to informational*

- Test out new checks without breaking the analysis

Post Processing - check.py

- Identify failed checks via ‘offenders’
- Custom checks on top of reports using DataExtract
- Combine filesystem image reports into Firmware report





Post Processing - Custom Checks

Compare data across filesystem images

- e.g. compare ‘data’ items from multiple reports

Analyze FileTree

- e.g., statistical analysis of file types

Example: Android Firmware

Disclaimer

- This is just an example!
- I searched the web for a **userdebug Android firmware** for this talk
- The aim is to demo the tool, not to find bugs

RockChip PX5

```
gs1_px30_8.1_ota(20181206).zip
```

source: https://www.aliexpress.com/item/Octa-core-9-RAM-2G-Android6-0-car-radio-dvd-gps-for-Toyota-Prado-2014-with/32806168855.html

Android OTA Update

check_ota.py will...

- unpack ota.zip, reconstruct sparse *.img files, and unpack boot.img

```
  boot_img/                          (unpacked boot.img)
           img_info
           kernel
           ramdisk/                  (unpacked ramdisk)
                   verity_key
                   ...
  boot.img                           (unpacked ota.zip)
system.img
vendor.img
   oem.img
 trust.img
 uboot.img
```

Configuration *(one for each filesystem image)*

system.toml

```
[GlobalConfig]
FsType = "extfs"
# enable SeLinux
FsTypeOptions = "selinux"
DigestImage = true

[GlobalFileChecks]
Suid = true
# run-as is a common suid binary
SuidWhiteList = ["/bin/runs-as"]
# enable SeLinux checks
SeLinuxLabel = true
# system is mounted read-only
WorldWrite = true
# UIDs and GIDs need to be adjusted for each device
Uids = [0,1000,1003,2000]
Gids = [0,1000,1003,2000]
BadFiles = ["/xbin/su", "/xbin/tcpdump"]
```

boot.toml

```
[GlobalConfig]
FsType = "dirfs"

[GlobalFileChecks]
Suid = true
SuidWhiteList = []
BadFiles = []
```

vendor.toml

```
[GlobalConfig]
FsType = "extfs"
# enable SeLinux
FsTypeOptions = "selinux"
DigestImage = true

[GlobalFileChecks]
Suid = true
SuidWhiteList = [""]
# enable SeLinux checks
SeLinuxLabel = true
# system is mounted read-only
WorldWrite = true
# UIDs and GIDs need to be adjusted for
Uids = [0,1000,1003,2000]
Gids = [0,1000,1003,2000]
BadFiles = []
```

system.toml

```
[GlobalConfig]
FsType = "extfs"
# enable SeLinux
FsTypeOptions = "selinux"
DigestImage = true

[GlobalFileChecks]
Suid = true
# run-as is a common suid binary
SuidWhiteList = ["/bin/runs-as"]
# enable SeLinux checks
SeLinuxLabel = true
# system is mounted read-only
WorldWrite = true
# UIDs and GIDs need to be adjusted
Uids = [0,1000,1003,2000]
Gids = [0,1000,1003,2000]
BadFiles = ["/xbin/su", "/xbin/tcpdump"]
```

```
[FileTreeCheck]
OldTreeFilePath = "system_filetree.json"
CheckPermsOwnerChange = true
CheckFileSize = false

[FilePathOwner."/etc"]
Uid = 0
Gid = 0

[FileContent."/etc/security/otacerts.zip"]
File = "/etc/security/otacerts.zip"
Digest = "d44ebdd80c1aa48565e626b9521b7bd484395a92865df29

# include checks for user builds
[Include."android_user_checks.toml"]

# ex
[Include."android_properties.toml"]
```

include checks for user builds (aka production build)

boot.toml

```
[GlobalConfig]
FsType = "dirfs"

[GlobalFileChecks]
Suid = true
SuidWhiteList = []
BadFiles = []

[FileTreeCheck]
OldTreeFilePath = "boot_filetree.json"
CheckPermsOwnerChange = false
CheckFileSize = false
SkipFileDigest = true

[FileContent."verity key"]
File = "/boot_img/ramdisk/verity_key"
Digest = "d10ada18b08ca1282c2b7ef7bf4b05fa05786de2061796cff530cfa5d898ec8e"

[Include."boot_img_checks.toml"]

[DataExtract."kernel_cmd_line"]
File = "/boot_img/img_info"
Regex = ".*cmd_line='(.+)'.*"
```

FsType = local directory (remember we had to unpack the boot.img)

android_user_build_checks.toml

Check “user” properties

- ro.build
- ro.secure
- ro.debuggable

```
# -- Android user build checks --
#
# Basic checks for a user (production) build.
# Checks cover: system.img

[FileContent."ro.build=user"]
File = "/build.prop"
Regex = ".*\\nro\\.build\\.type=user\n.*"
Desc = "build type must be: user"

[FileContent."ro.secure=1"]
File = "/etc/prop.default"
Regex = ".*\\nro\\.secure=1.*"
Desc = "ro.secure must be 1"

[FileContent."ro.debuggable=0"]
File = "/etc/prop.default"
Regex = ".*\\nro\\.debuggable=0.*"
Desc = "ro.debuggable must be 0"
```

android_properties.toml

Extract properties ro.build...

- type
- flavor
- security_patch

```
# - build.prop -

[DataExtract."ro.build.type__1"]
File = "/build.prop"
RegEx = ".*\\nro\\.build\\.type=(\\S+)\\n.*"

[DataExtract."ro.build.tags__1"]
File = "/build.prop"
RegEx = ".*\\nro\\.build\\.tags=(\\S+)\\n.*"

[DataExtract."ro.build.flavor__1"]
File = "/build.prop"
RegEx = ".*\\nro\\.build\\.flavor=(\\S+)\\n.*"

[DataExtract."ro.build.id__1"]
File = "/build.prop"
RegEx = ".*\\nro\\.build\\.id=(\\S+)\\n.*"

[DataExtract."ro.build.version.security_patch__1"]
File = "/build.prop"
RegEx = ".*\\nro\\.build\\.version\\.security_patch=(\\S+)\\n.*"
```

Running it...

check_ota.py does all the leg work for you...

- unzip the OTA, reassemble the filesystem images, unpack boot.img
- run FwAnalyzer
- inspect the individual filesystem reports and assemble the final report

```
$ check_ota.py --ota unpacked --cfg-path demo/user --targets
"system,boot,vendor" --cfg-include-path demo --report demo_user.json

report written to: demo_user.json
OTA Check Failed
$
```

Report

Check “Status”

Image reports

- boot
- system
- vendor


```
{
  "Firmware": "gs1_px30_8.1_ota(20181206).zip",
  "Status": false,
  "boot": {
    "CurrentFileTreePath": "/home/cmulliner/research/talk/demo/user/boot_filetree.json.new",
    "Data": {
      "kernel_cmd_line": "buildvariant=userdebug"
    },
    "FSType": "dirfs",
    "ImageName": "/home/cmulliner/research/talk/unpacked/",
    "Offenders": { ...
    },
    "OldFileTreePath": "/home/cmulliner/research/talk/demo/user/boot_filetree.json"
  },
  "system": {
    "CurrentFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json.new",
    "Data": { ...
    },
    "FSType": "extfs",
    "ImageDigest": "7896ae177e5504ca6d9706f2f896da483e423d68cccc398948fb8fa3f8751ccb",
    "ImageName": "/home/cmulliner/research/talk/unpacked/system.img",
    "Offenders": { ...
    },
    "OldFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json"
  },
  "vendor": {
    "CurrentFileTreePath": "/home/cmulliner/research/talk/demo/user/vendor_filetree.json.new",
    "Data": { ...
    },
    "FSType": "extfs",
    "ImageDigest": "eb190644289a1af3765fea843573c203329a6c44e917bbb5804701fc76d7261b",
    "ImageName": "/home/cmulliner/research/talk/unpacked/vendor.img",
    "Offenders": { ...
    },
    "OldFileTreePath": "/home/cmulliner/research/talk/demo/user/vendor_filetree.json"
  }
}
```

Offenders

ro.build != user

bad otacerts.zip

SUID files

illegal files

bad UIDs


```
{
  "Firmware": "gs1_px30_8.1_ota(20181206).zip",
  "Status": false,
  "boot": { ...
  },
  "system": {
    "CurrentFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json.new",
    "Data": { ...
    },
    "FSType": "extfs",
    "ImageDigest": "7896ae177e5504ca6d9706f2f896da483e423d68cccc398948fb8fa3f8751ccb",
    "ImageName": "/home/cmulliner/research/talk/unpacked/system.img",
    "Offenders": {
      "/bin/logd": [
        "File Uid not allowed, Uid = 1036",
        "File Gid not allowed, Gid = 1036"
      ],
      "/build.prop": [
        "RegEx check failed, for: ro.build=user : build type must be: user"
      ],
      "/etc/prop.default": [
        "RegEx check failed, for: ro.debuggable=0 : ro.debuggable must be 0"
      ],
      "/etc/security/otacerts.zip": [
        "Digest (sha256) did not match found = a3da41fb805fd6721d2f81a1b0c41ebe2e039d2e6604440ab5188f5ed6244964 shou
        d44ebdd80c1aa48565e626b9521b7bd484395a92865df293a3aac8f911f0bdab. /etc/security/otacerts.zip :  "
      ],
      "/xbin/procmem": [
        "File is SUID, not allowed"
      ],
      "/xbin/su": [
        "File is SUID, not allowed",
        "File not allowed"
      ],
      "/xbin/tcpdump": [
        "File not allowed"
      ]
    },
    "OldFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json"
  },
  "vendor": { ...
  }
}
```

Data


```
{
  "Firmware": "gs1_px30_8.1_ota(20181206).zip",
  "Status": false,
  "boot": { ...
  },
  "system": {
    "CurrentFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json.new",
    "Data": {
      "ro.bootimage.build.date": "Thu Dec 6 10:57:35 CST 2018",
      "ro.bootimage.build.fingerprint": "rockchip/rk3326_mid/rk3326_mid:8.1.0/OPM6.171019.030.E1/hct12061057
      "ro.build.date": "Thu Dec  6 10:57:35 CST 2018",
      "ro.build.flavor": "rk3326_mid-userdebug",
      "ro.build.id": "OPM6.171019.030.E1",
      "ro.build.tags": "test-keys",
      "ro.build.type": "userdebug",
      "ro.build.version.codename": "REL",
      "ro.build.version.incremental": "eng.hct.20181206.105735",
      "ro.build.version.release": "8.1.0",
      "ro.build.version.security_patch": "2018-09-05",
      "ro.debuggable": "1",
      "ro.product.device": "rk3326_mid",
      "ro.product.name": "rk3326_mid"
    },
    "FSType": "extfs",
    "ImageDigest": "7896ae177e5504ca6d9706f2f896da483e423d68cccc398948fb8fa3f8751ccb",
    "ImageName": "/home/cmulliner/research/talk/unpacked/system.img",
    "Offenders": { ...
    },
    "OldFileTreePath": "/home/cmulliner/research/talk/demo/user/system_filetree.json"
  },
  "vendor": { ...
  }
}
```

Result

‘userdebug’ OTA firmware failed production checks! *(as it should)*

Visibility - with one look we can easily tell

- Failed checks (offenders)

- Version and security patch level
- Build Flavor
- Security properties (ro.debuggable, …)

Deploying FwAnalyzer

- Development
  - Find issues early and have developers fix them as they build the product
- Gate for Production Signing
  - Make sure you only sign “good” production builds
- Acceptance Testing
  - You buy a custom device from a supplier and want to ensure the quality of the firmware
  - You create rules to check firmware updates as you receive them
- Vendor/Supplier Assessment
  - You buy Commercial off-the-shelf devices and need to assess them
  - (FwAnalyzer as one tool in the process)

FwAnalyzer during Development

Goal: prevent development based on bad/incomplete information

- Developers want to do the right thing but might not know security

→ Provide feedback about security during development

FwAnalyzer as part of Tests

- `make test` also invokes FwAnalyzer
- run FwAnalyzer in CI

make developers aware about file permissions, security labels, SUID binaries, ...

FwAnalyzer in CI: SELinux

SELinux breaks developers assumptions about file access

- Missing or wrong label will break file access

Lots and lots of frustration

- Security is blamed for breaking everything (why does reading file X fail???)

FwAnalyzer flags unlabeled files

```
[GlobalFileChecks]
SeLinuxLabel = true
```

FwAnalyzer in CI: File Permissions and Ownership

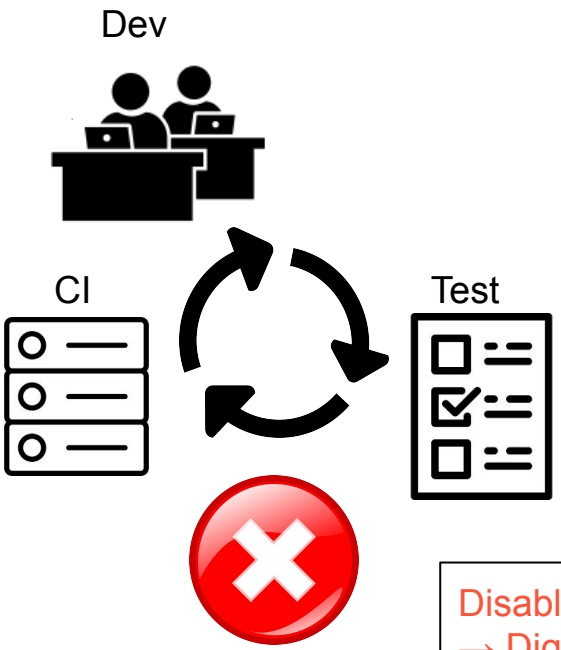
File Permissions and Ownership are easily overlooked

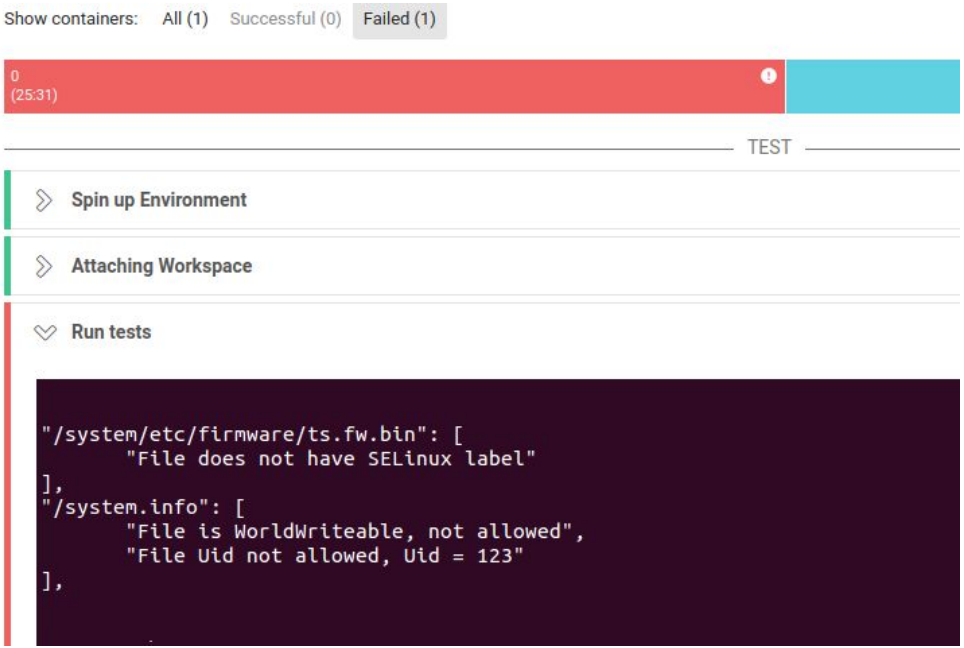
- Platform ONLY needs to READ the file → everything works, right?

FwAnalyzers flags World Writable files & check file owner user/group

```
[GlobalFileChecks]
WorldWrite = true
Uids = [0,1001,1002]
Gids = [0,1001,1002]
```

FwAnalyzer in CI





Disable “slower” checks in the CI config of FwAnalyzer
→ DigestImage, FileTreeChecks

Production Signing

Production signed firmware will run on every device out in the field

- Producing “insecure” production signed firmware might cause huge damages
  - Highly dependent on your risk model and other security features such as rollback protection
  - You never ever want to production sign a development build!

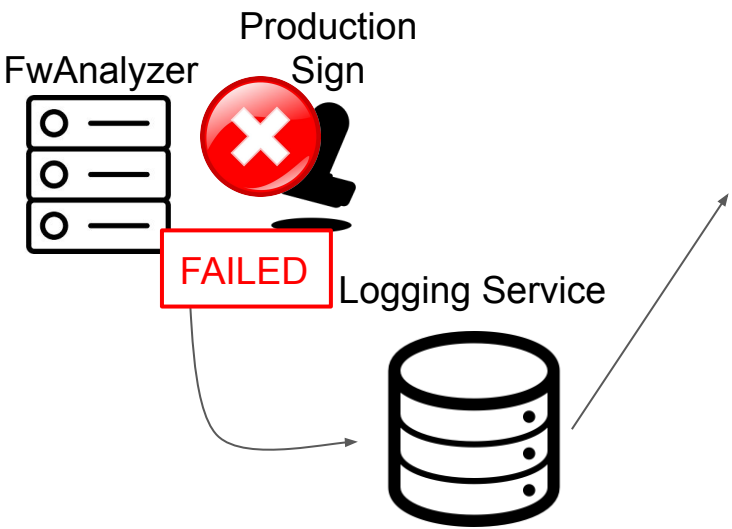
Production Signing should ONLY happen after development and testing is done

→ The final stamp of approval before shipping

Production Checks

- BuildType Information
  - eng vs production
- Public keys/certs (e.g. used for software update)
  - ensure they are the production version
- System hardening features
  - secure boot, dm-verity configured and enabled, hardened kernel config, stripped binaries, ...
- Development/Debug/Test Binaries
  - strace, tcpdump, sshd, … (highly OS and product dependent)
- Production Config Files
  - hardened firewall config, debug/test user accounts

FwAnalyzer Gating Production Signing




```
{
  "Firmware": "ota.zip",
  "Status": false,
  "boot": {
    "Data": {
      "kernel_cmd_line": "buildvariant=userdebug"
    },
    "FSType": "dirfs",
    "ImageName": "unpacked/boot_img",
    "Offenders": {
      "/boot_img/img_info": [
        "RegEx check failed, for: selinux=enforcing (ramdisk) : selinux should be set to enforcing",
        "RegEx check failed, for: buildvariant=user (ramdisk) : buildvariant must be: user"
      ]
    }
  },
  "system": {
    "Data": {
      "ro.build.flavor": "xxx-userdebug",
      "ro.build.tags": "release-keys",
      "ro.build.type": "userdebug",
      "ro.build.version.release": "8.1.0",
      "ro.build.version.security_patch": "2018-09-05",
      "ro.debuggable": "1",
    },
    "FSType": "extfs",
    "ImageName": "unpacked/system.img",
    "Offenders": {
      "/build.prop": [
        "RegEx check failed, for: ro.build=user : build type must be: user"
      ],
      "/etc/prop.default": [
        "RegEx check failed, for: ro.debuggable=0 : ro.debuggable must be 0"
      ],
      "/xbin/su": [
        "File is SUID, not allowed",
        "File not allowed"
      ]
    },
  }
}
```


multiple checks failed

Acceptance Testing

Device manufactured by 3rd party supplier

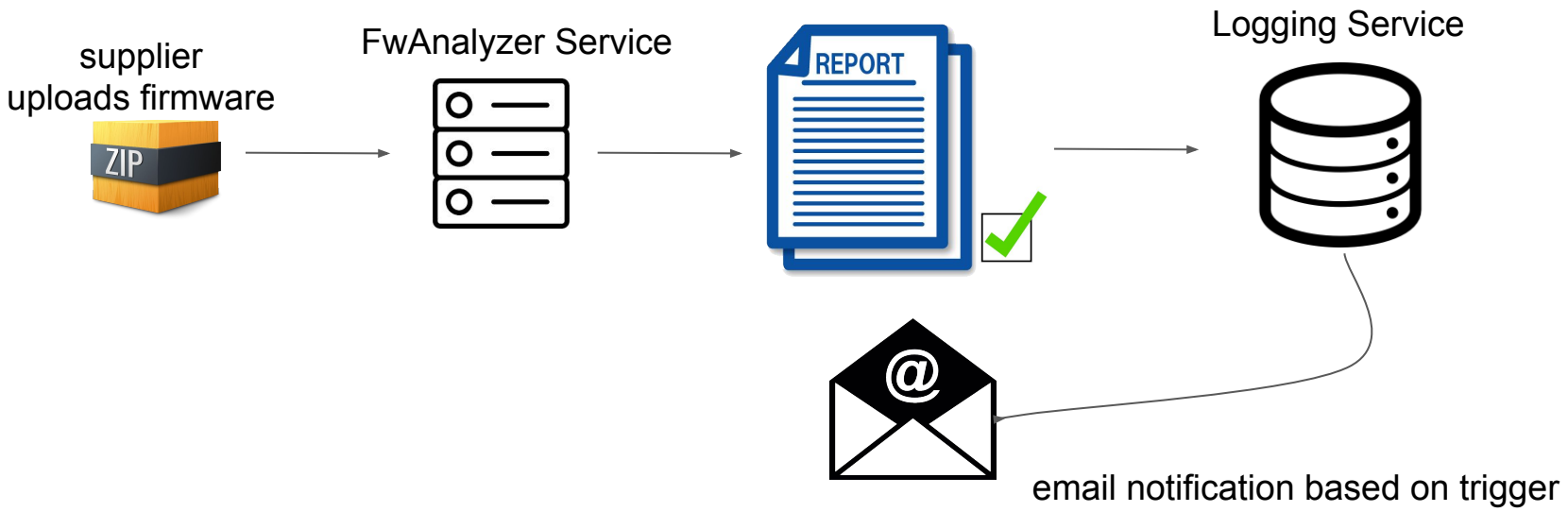
- You spent time to figure out how everything works
- Went back and forth with supplier until “every works as expected”

Create FwAnalyzer configuration to capture known good state

- Check new firmware revisions using configuration with known “good state”
- Include as much data as possible in the report (DataExtract)

catch changes that later might bite you in the...

FwAnalyzer as part of Acceptance Testing



Acceptance Testing cont.

New firmware revision, do changes...

- violate any of your rules?
- match up with the change log from the supplier?

Detailed report provides good visibility into firmware revision.

Vendor Security Assessment

Asses Commercial Off-the-Shelf devices

Quickly find interesting things...

- SUID binaries, world writable files, ...
- Cryptographic material, account names and password hashes

FileTreeCheck...

- Discover changes between firmware versions

FwAnalyzer just as one step in the process

Post Post Processing

Pump FwAnalyzer JSON output into your favorite log service

Implement post processing checks based on ...

- Filenames
- Extracted data

Configure Searches/Triggers

- Feed into Alerting, Reporting, and Ticketing

FwAnalyzer for Your Device

Tasks:

- **unpack YOUR firmware format**
- **create FwAnalyzer config files**
- custom checks & post processing of results

check.py, implement:

- firmware unpacking
- report post processing and custom complex checks

FwAnalyzer for Your Device: Config Files

Create a config file for each FileSystem image you unpacked

- select FsType that matches your filesystem image
- use DirFs to handle files extracted from BLOBs such as boot.img

Rules:

- start with GlobalFileChecks
- slowly add specific checks (use InformationalOnly to test)
- enable FileTree checks to monitor revision changes
- move checks to rule library so you can share them across devices

Rule Library

```
[Include."common_checks.toml"]
```

Write checks and DataExtract rules once...

- Common rules for dev and prod
- Multiple similar devices

```
$ ls devices/android
android_properties.toml
android_user_build_checks.toml

$ ls devices/generic/linux
mounts.toml
path_owners.toml
```

Similarly for scripts….

Open source and share rule library with the community!

Future Work

FwAnalyzer

- FileSystems : support more filesystem types

Scripts

- Kernel config : check for debug features, etc...
- checksec.sh* : check security features of executables (ASLR, DEP, ...)
- APK checker : use APK security checking tool

*https://github.com/slimm609/checksec.sh

Conclusions

FwAnalyzer

- Deployed in CI for one big project → more to follow soon
- Gates production signing for multiple devices

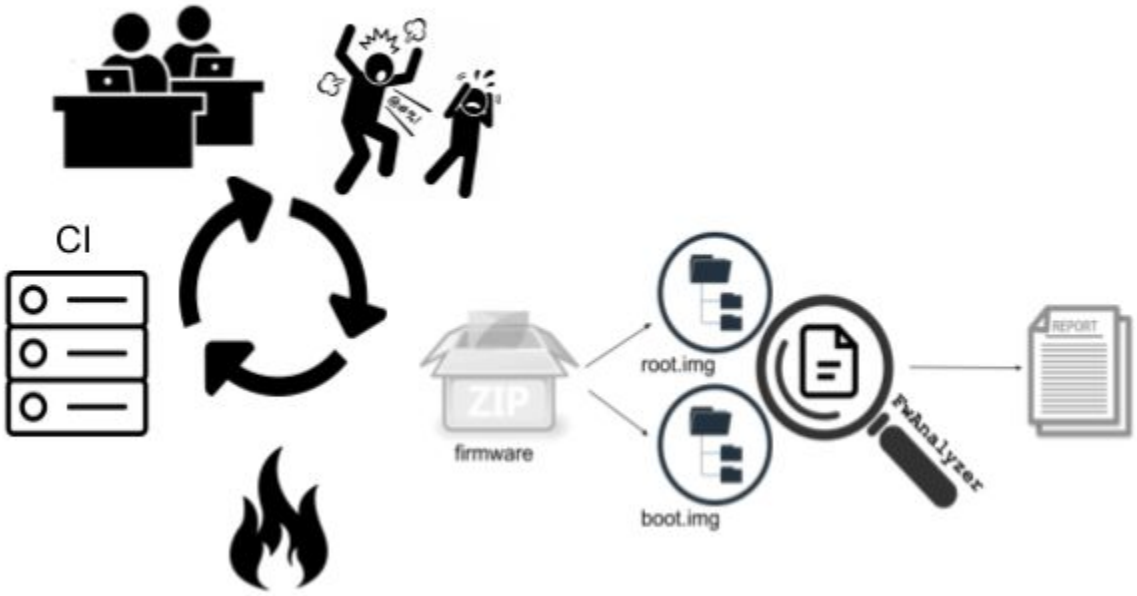
All of this is just one piece of the puzzle!

- You still have to: build security controls, audit code, …

Continuous Security Analysis has shown value

- Caught potential showstoppers … one could have been expensive

Continuous Automated Firmware Security Analysis



Continuous Automated Firmware Security Analysis

Won’t easily miss security relevant changes

Automated feedback during development

Confidence and peace of mind

Thanks & Credits

Code Contributors

- **Jon Larimer and Graziano Misuraca**

General Support

- Chris Valasek, Tim Piastrelli, and others Cruisers

Discussions and feedback

- Various ‘unnamed’ people

Public Release

Today FwAnalyzer is Open Source

We have a blog post out at: https://medium.com/cruise

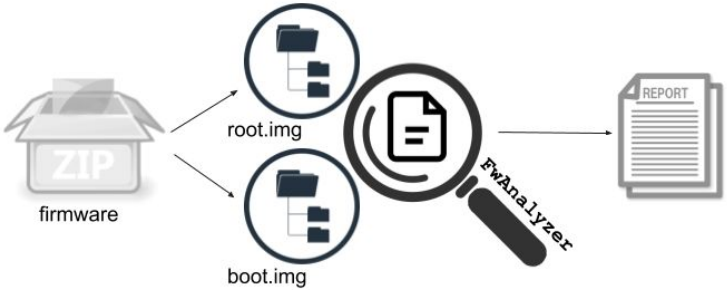
Source on GitHub: https://github.com/cruise-automation/fwanalyzer

Release includes:

- Config example and rule library
- Lots of unit and integration tests
- Full end-to-end check for Android OTA files (unpacking & configs)

END - thank you for your time!

Questions?



www.FwAnalyzer.io

...or bug me on Twitter at: ***@collinrm***