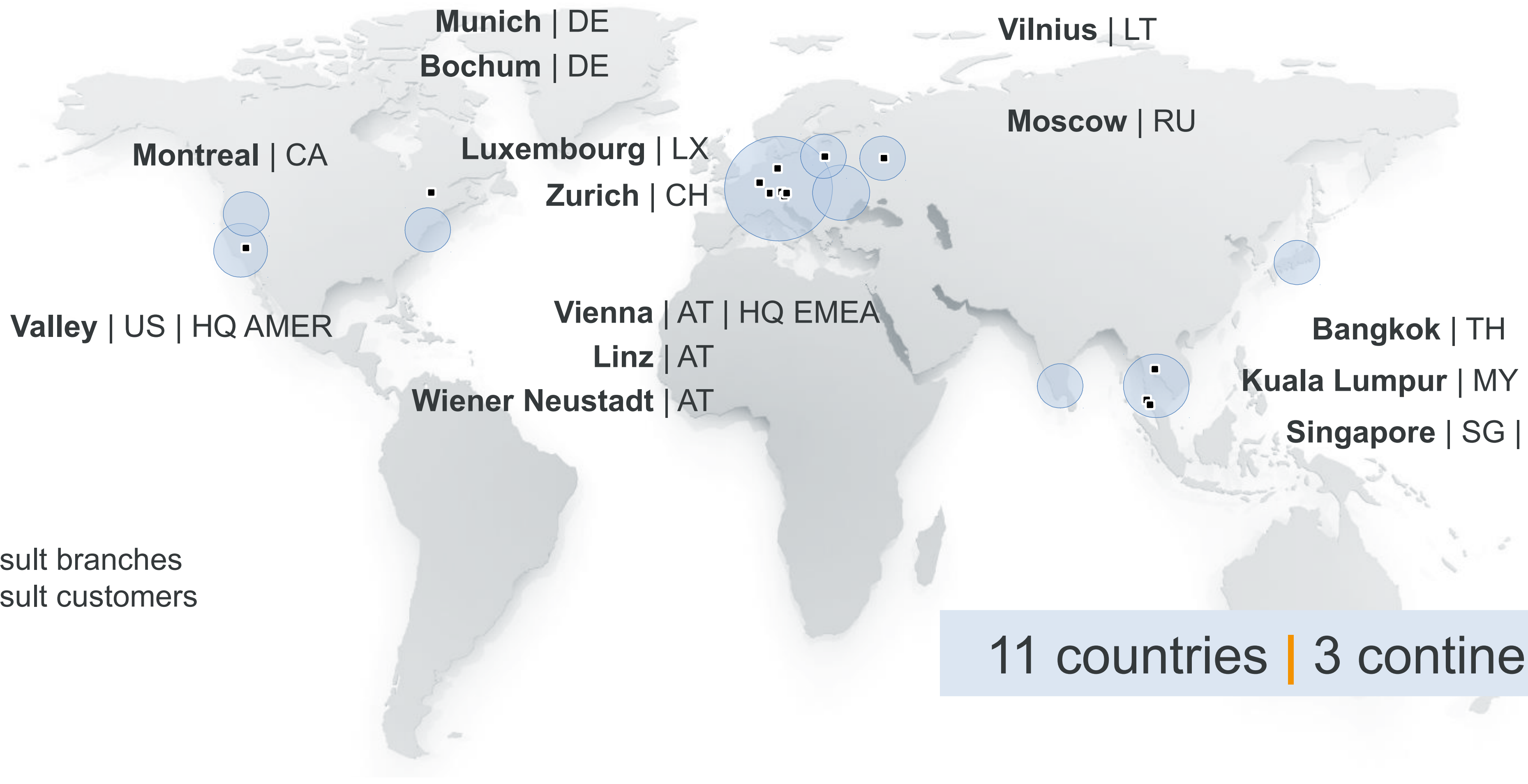
#lockdownlivestream

Army of Undead – Tailored Firmware Emulation

Researched@ FAKULTÄT FÜR INFORMATIK Faculty of Informatics

Case studies done@ SEC Consult



Berlin| DE
Munich | DE
Bochum | DE

Vilnius | LT

Moscow | RU

Montreal | CA

Luxembourg | LX
Zurich | CH

Silicon Valley | US | HQ AMER

Vienna | AT | HQ EMEA
Linz | AT
Wiener Neustadt | AT

Bangkok | TH
Kuala Lumpur | MY
Singapore | SG | HQ APAC

■ SEC Consult branches
● SEC Consult customers

11 countries | 3 continents

About me.

Thomas Weber
t.weber@sec-consult.com

SEC Consult Unternehmensberatung GmbH
Leopold-Ungar-Platz 2/3/3
1190 Vienna, AUSTRIA

www.sec-consult.com

Outline

- What? Why?
- Nowadays Firmware Development
- Detection of Essential Parts
- Preparing Fake Images
- Demo Time
- Monitoring and Debugging
- Scaled Study
- Vulnerabilities
- Conclusion and Further Work

What? // Expectations

The talk is about …

- **Linux based** firmware **with EABI**, no complete RTOS like VXworks. Windows can be covered with a similar approach but was not implemented in this prototype.

What? // Expectations

The talk is about …

- **Linux based** firmware **with EABI**, no complete RTOS like VXworks. Windows can be covered with a similar approach but was not implemented in this prototype.
- dynamic analysis, which means: libraries, networked services and known exploits.

What? // Expectations

The talk is about …

- **Linux based** firmware **with EABI**, no complete RTOS like VXworks. Windows can be covered with a similar approach but was not implemented in this prototype.
- dynamic analysis, which means: libraries, networked services and known exploits.
- a feasibility study about a workflow to realize simple firmware emulation with open source tools. It is not a ready-to-use tool!

What? // Expectations

The talk is about …

- **Linux based** firmware **with EABI**, no complete RTOS like VXworks. Windows can be covered with a similar approach but was not implemented in this prototype.
- dynamic analysis, which means: libraries, networked services and known exploits.
- a feasibility study about a workflow to realize simple firmware emulation with open source tools. It is not a ready-to-use tool!
- more than 170 firmware images that were emulatable across more than 49 vendors.

What? // Expectations

The talk is about …

- **Linux based** firmware **with EABI**, no complete RTOS like VXworks. Windows can be covered with a similar approach but was not implemented in this prototype.
- dynamic analysis, which means: libraries, networked services and known exploits.
- a feasibility study about a workflow to realize simple firmware emulation with open source tools. It is not a ready-to-use tool!
- more than 170 firmware images that were emulatable across more than 49 vendors.
- advisories that have been released, based on emulated devices only.

What? // Expectations

The talk is about …

- **Linux based** firmware **with EABI**, no complete RTOS like VXworks. Windows can be covered with a similar approach but was not implemented in this prototype.
- dynamic analysis, which means: libraries, networked services and known exploits.
- a feasibility study about a workflow to realize simple firmware emulation with open source tools. It is not a ready-to-use tool!
- more than 170 firmware images that were emulatable across more than 49 vendors.
- advisories that have been released, based on emulated devices only.
- a project that was started in January 2017.

Why? // Vantage Point

Because …

- it can be used to do remote testing with frame-conditions

Why? // Vantage Point

Because …

- it can be used to do remote testing with frame-conditions
- it was helpful for a lot of research/customer-projects

Why? // Vantage Point

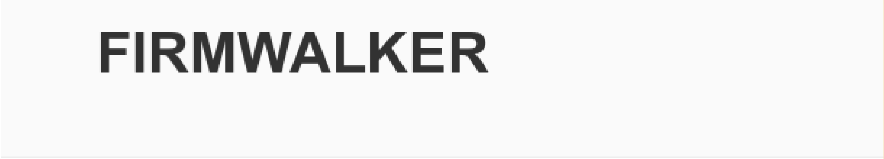
Because …

- it can be used to do remote testing with frame-conditions
- it was helpful for a lot of research/customer-projects
- dynamic analysis is currently limited to few architectures like ARM and MIPS

Why? // Vantage Point

Because …

- it can be used to do remote testing with frame-conditions
- it was helpful for a lot of research/customer-projects
- dynamic analysis is currently limited to few architectures like ARM and MIPS
- static analysis is well covered by many (non-)commercial solutions


FIRMWALKER

Why? // Vantage Point

Because …

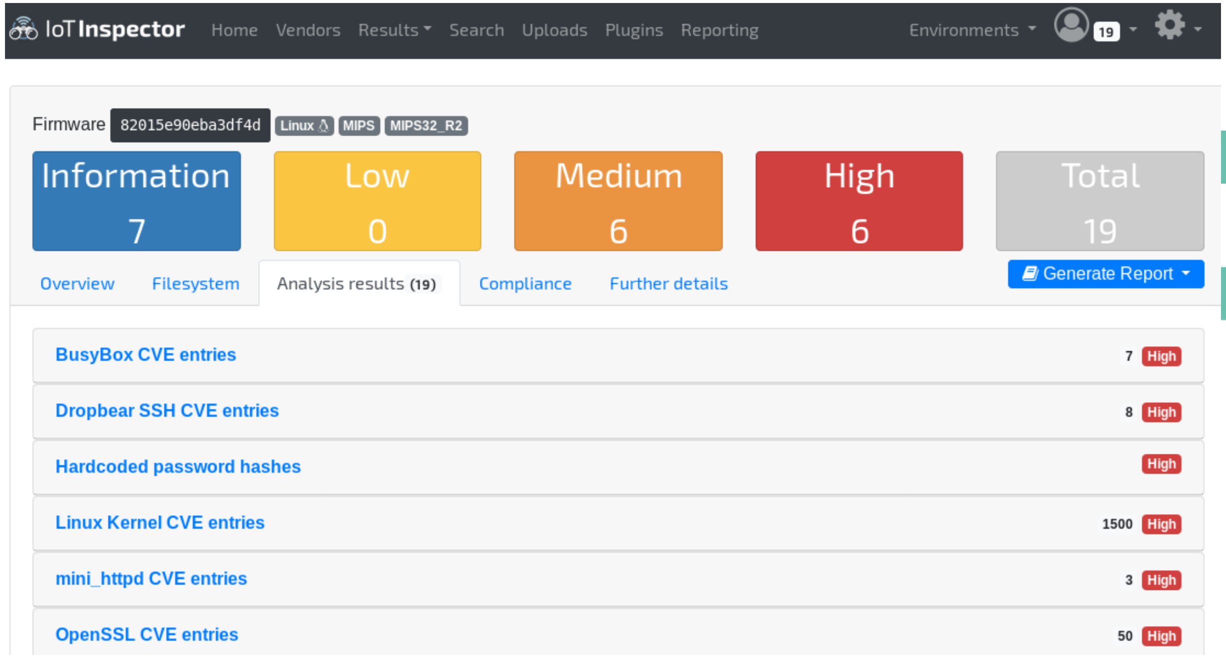
- it can be used to do remote testing with frame-conditions
- it was helpful for a lot of research/customer-projects
- dynamic analysis is currently limited to few architectures like ARM and MIPS
- static analysis is well covered by many (non-)commercial solutions

FIRMWALKER

Why? // Vantage Point

Because …

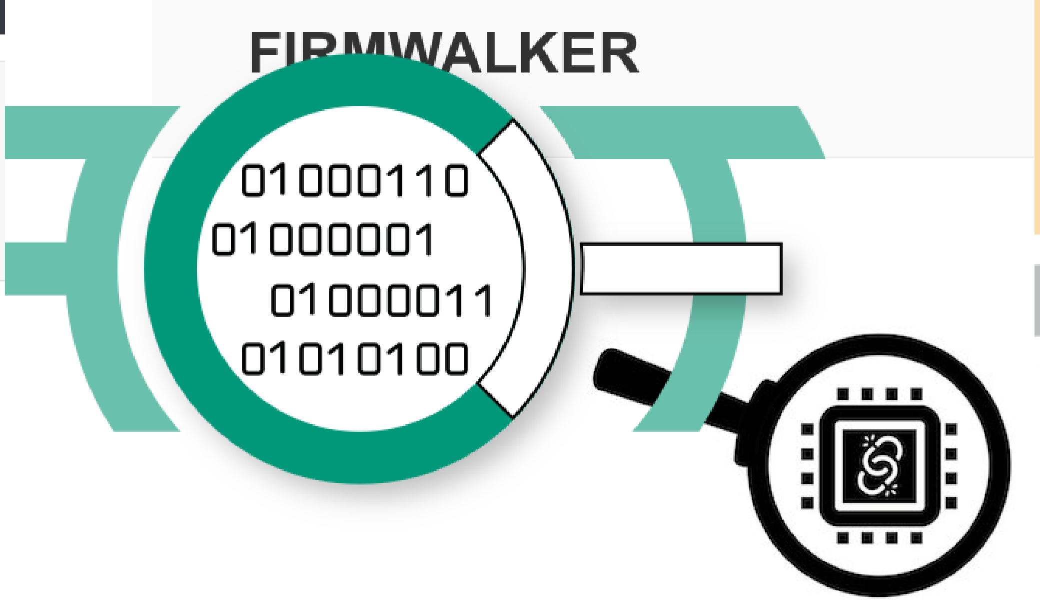
- it can be used to do remote testing with frame-conditions
- it was helpful for a lot of research/customer-projects
- dynamic analysis is currently limited to few architectures like ARM and MIPS
- static analysis is well covered by many (non-)commercial solutions



Firmware 82015e90eba3df4d Linux MIPS MIPS32_R2

Information	Low	Medium	High	Total
7	0	6	6	19

BusyBox CVE entries	7	High
Dropbear SSH CVE entries	8	High
Hardcoded password hashes		High
Linux Kernel CVE entries	1500	High
mini_httpd CVE entries	3	High
OpenSSL CVE entries	50	High



Nowadays Firmware Development

Bear in mind that nobody wants to do work twice! Buildroot is a very handy tool.

Nowadays Firmware Development

Bear in mind that nobody wants to do work twice! Buildroot is a very handy tool.

The most projects regarding development of embedded devices are based on one or multiple software/hardware development kits.

Nowadays Firmware Development

Bear in mind that nobody wants to do work twice! Buildroot is a very handy tool.

The most projects regarding development of embedded devices are based on one or multiple software/hardware development kits.

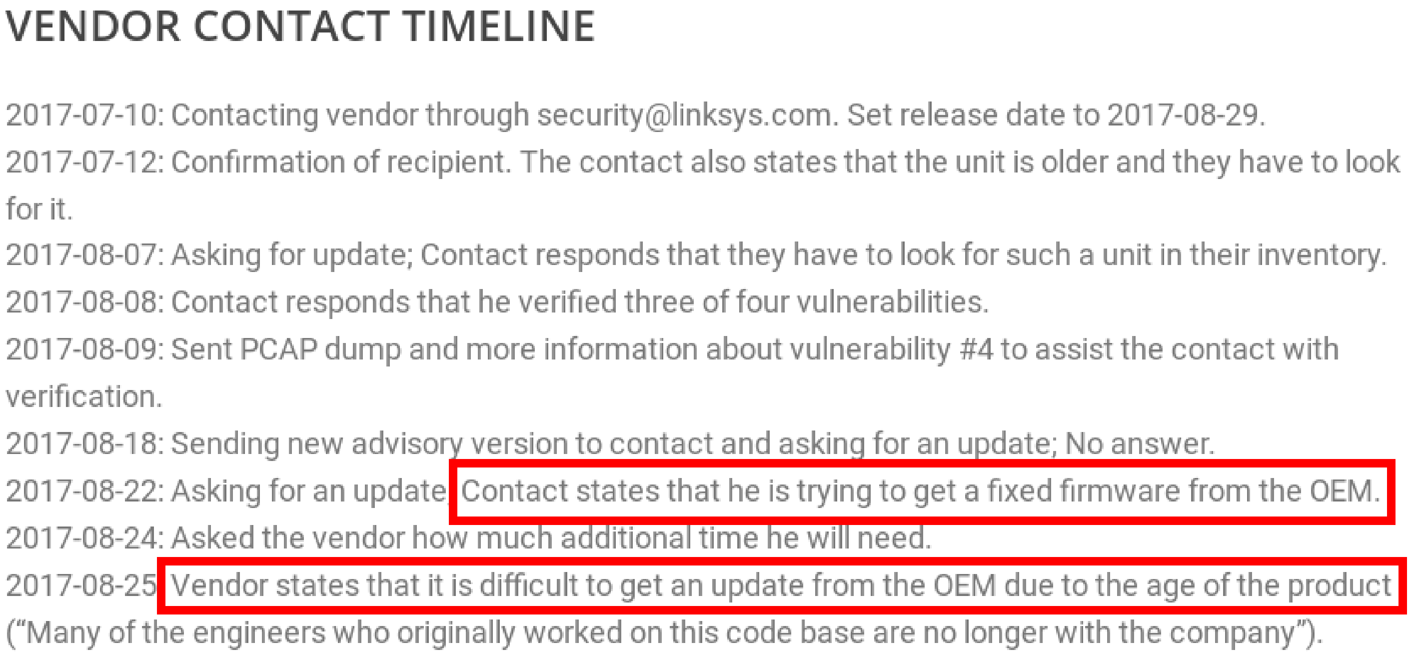
End-of-life, time-to-market and outsourcing are prominent buzzwords/-phrases.

Nowadays Firmware Development

Bear in mind that nobody wants to do work twice! Buildroot is a very handy tool.

The most projects regarding development of embedded devices are based on one or multiple software/hardware development kits.

End-of-life, time-to-market and outsourcing are prominent buzzwords/-phrases.

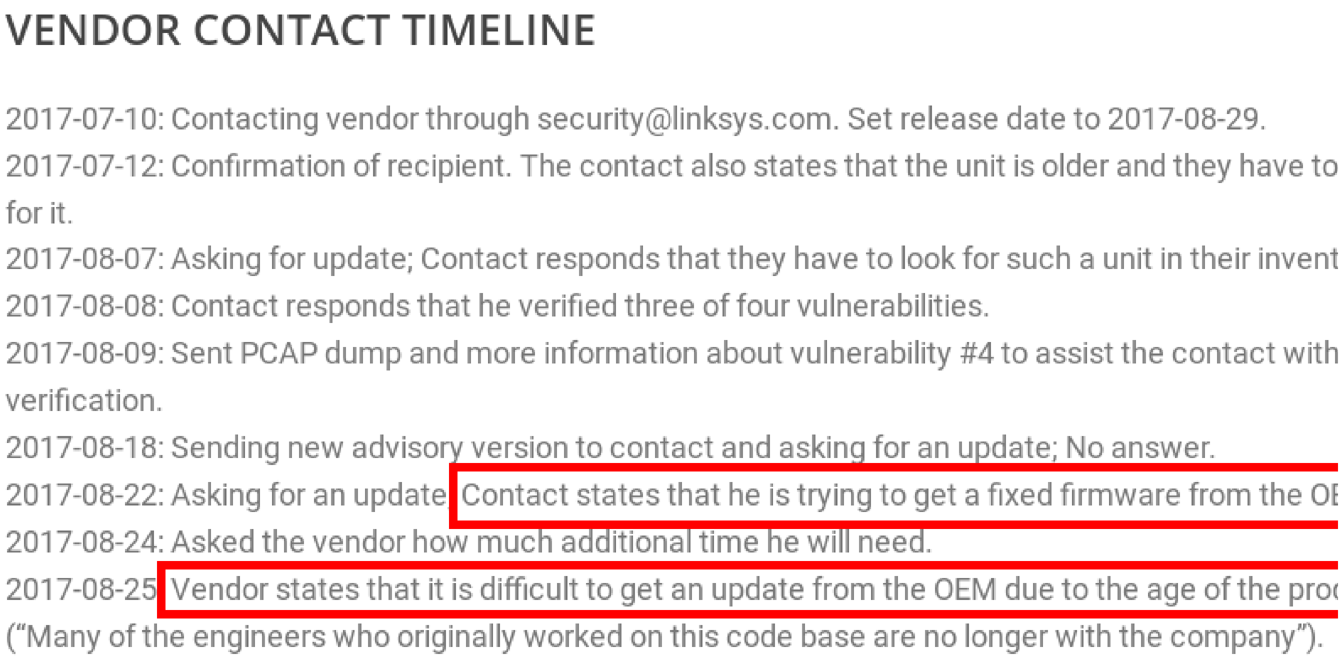


VENDOR CONTACT TIMELINE

2017-07-10: Contacting vendor through security@linksys.com. Set release date to 2017-08-29.
2017-07-12: Confirmation of recipient. The contact also states that the unit is older and they have to look for it.
2017-08-07: Asking for update; Contact responds that they have to look for such a unit in their inventory.
2017-08-08: Contact responds that he verified three of four vulnerabilities.
2017-08-09: Sent PCAP dump and more information about vulnerability #4 to assist the contact with verification.
2017-08-18: Sending new advisory version to contact and asking for an update; No answer.
2017-08-22: Asking for an update. Contact states that he is trying to get a fixed firmware from the OEM.
2017-08-24: Asked the vendor how much additional time he will need.
2017-08-25: Vendor states that it is difficult to get an update from the OEM due to the age of the product ("Many of the engineers who originally worked on this code base are no longer with the company").

Nowadays Firmware Development

- Bear in mind that nobody wants to do work t
- The most projects regarding development of multiple software/hardware development kits
- End-of-life, time-to-market and outsourcing a



VENDOR CONTACT TIMELINE

2017-07-10: Contacting vendor through security@linksys.com. Set release date to 2017-08-29.

2017-07-12: Confirmation of recipient. The contact also states that the unit is older and they have to for it.

2017-08-07: Asking for update; Contact responds that they have to look for such a unit in their invent

2017-08-08: Contact responds that he verified three of four vulnerabilities.

2017-08-09: Sent PCAP dump and more information about vulnerability #4 to assist the contact with verification.

2017-08-18: Sending new advisory version to contact and asking for an update; No answer.

2017-08-22: Asking for an update. Contact states that he is trying to get a fixed firmware from the OE

2017-08-24: Asked the vendor how much additional time he will need.

2017-08-25: Vendor states that it is difficult to get an update from the OEM due to the age of the pro ("Many of the engineers who originally worked on this code base are no longer with the company").

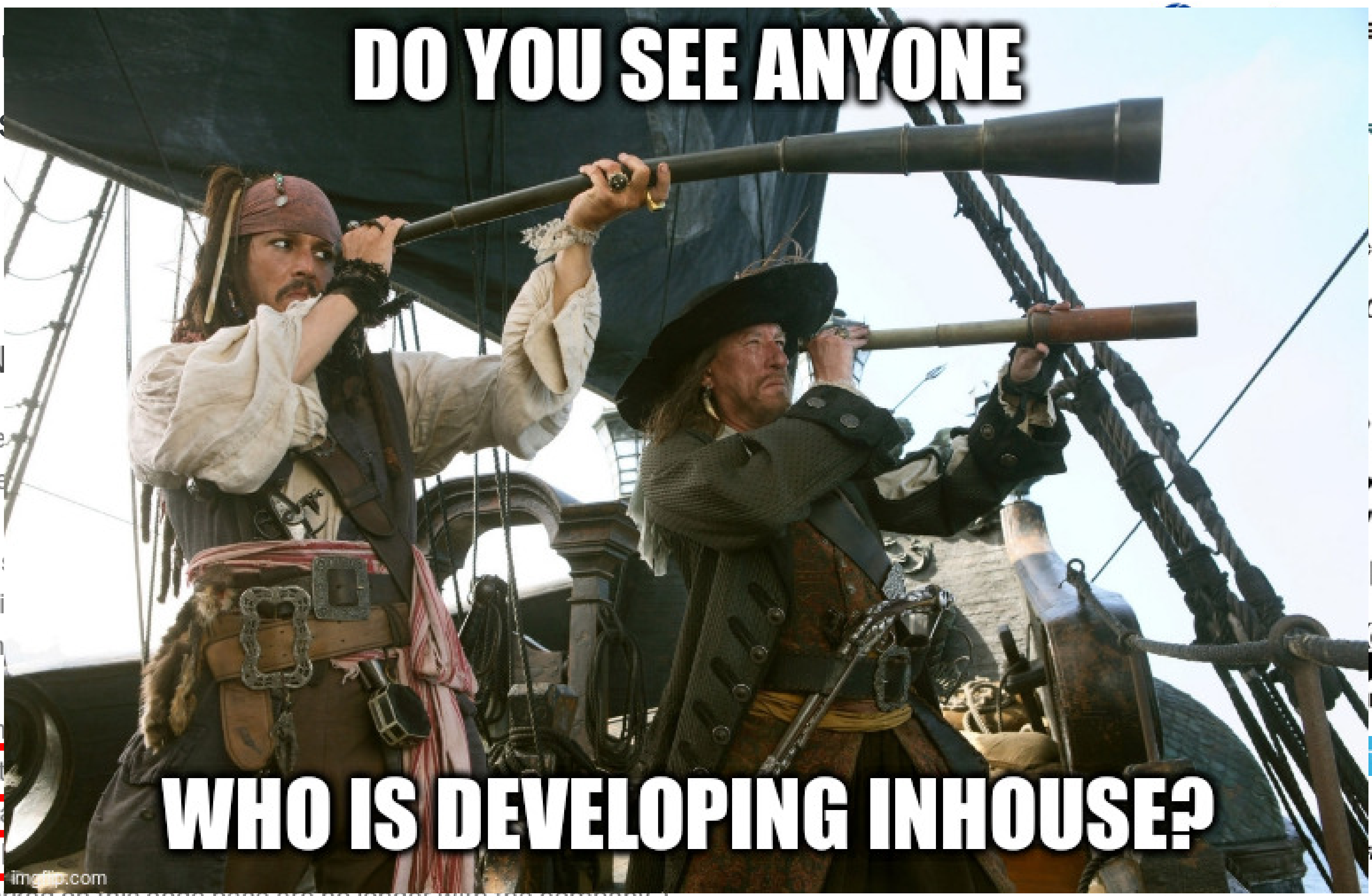
Xiongmai OEM Vendors

Nowadays Firmware Development

Bear in

The mos
multiple

End-of-l



VENDOR CONTACT TIMELIN

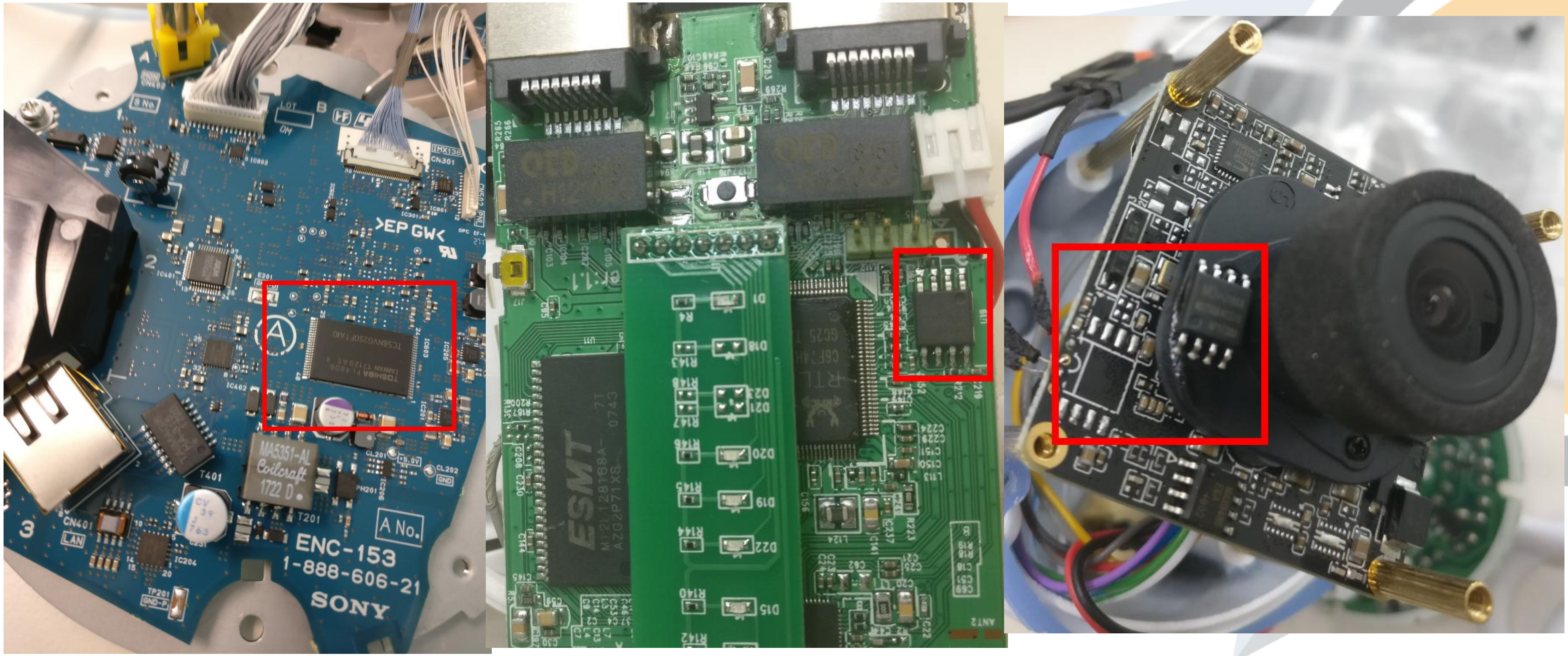
2017-07-10: Contacting vendor through se
2017-07-12: Confirmation of recipient. The
for it.
2017-08-07: Asking for update; Contact re
2017-08-08: Contact responds that he veri
2017-08-09: Sent PCAP dump and more in
verification.
2017-08-18: Sending new advisory version
2017-08-22: Asking for an update. Contact
2017-08-24: Asked the vendor how much
2017-08-25: Vendor states that it is difficu
("Many of the engineers who originally worked on this code base are no longer with the company").

Xiongmai OEM Vendors

Abowone
ANRAN
ASECAM
Autoeye
Azishn
A-ZONE
BESDER
RO
CATO CCTV
DEFEWAY
REON
Furi
Gadinan
NER
Jennov
JOOAN
OON
KONLEN
KOPDA
迈
nology
BUL
MISECU
NexTrend
l film
Susikum
smar
age
TECHEGE
TianAnXun
EQLO
Westshine
WISTINO
Witrue
WNK
WOFEA
WOSHIJIA
WUSONGLUSAN
信安讯
XinAnX
XLoongX
YUNSYE
zclèvér
ZILNK
Intelligent Electronic
Zjuxin
zmodo
JF TECH

Nowadays Firmware Development – Storage

In most cases, a small external flash chip or a SD-card is used.



Nowadays Firmware Development – Storage

In most cases, a small external flash chip or a SD-card is used.

These chips are usually SPI, NAND or NOR flash memory ICs.

Nowadays Firmware Development – Storage

In most cases, a small external flash chip or a SD-card is used.

These chips are usually SPI, NAND or NOR flash memory ICs.

They are either directly flashed with a programmer or written via JTAG or another debug-interface.

Nowadays Firmware Development – Storage

In most cases, a small external flash chip or a SD-card is used.

These chips are usually SPI, NAND or NOR flash memory ICs.

They are either directly flashed with a programmer or written via JTAG or another debug-interface.

A part of this memory is also used to store configurations like usernames, PINs etc. which is called NVRAM (Non-Volatile RAM). It used to be a separate IC.

Nowadays Firmware Development – Distribution / Device Upload

Distribution:

- Web-sites, FTP-servers, or as physical mail.
- Some vendors also use push-messages in the web-interface of the device.

Nowadays Firmware Development – Distribution / Device Upload

Distribution:

- Web-sites, FTP-servers, or as physical mail.
- Some vendors also use push-messages in the web-interface of the device.

Upload:

- Web-interface
- USB stick / SD-card
- TFTP / FTP
- Via an external programmer (JTAG)

Nowadays Firmware Development – Extract Firmware from Devices

Extracting Firmware can be done in by:

- JTAG/ISP/SWD programmers – this can be locked for some chips!
- Chip-off techniques – remove the flash chip and dump it directly. Have a look at our SEC Xtractor project (https://github.com/sec-consult/)
- Sniffing – can be done for a broad range of serial/parallel communication interfaces. Most prominent example is SPI.
- Side-Channel attacks – Glitching can lead to malfunctions for instructions. In specific cases, a UART interface can be abused to print out the whole content of the firmware.
- Microscopy – by using a SEM, an internal flash memory can be dumped in an optical way. Other ways are possible with microsurgery.

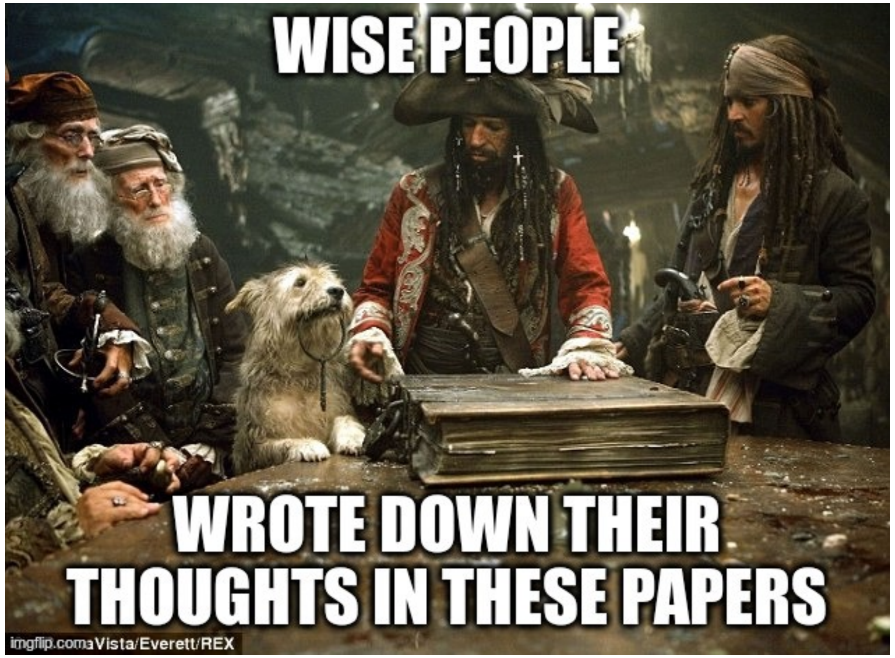
...and much more.

All Beginnings are Difficult

First of all, past publications about multi-arch firmware emulation were studied:

- Automated Dynamic Firmware Analysis at Scale:A Case Study on Embedded Web Interfaces (Costin et al.) – standard Debian Images / `chroot` in the target firmware
- FIRMADYNE (Chen et al.) – modified kernels with `musl-libc` / target firmware file system is directly used

Both projects were mostly covering ARM(EL), MIPS(BE/EL)



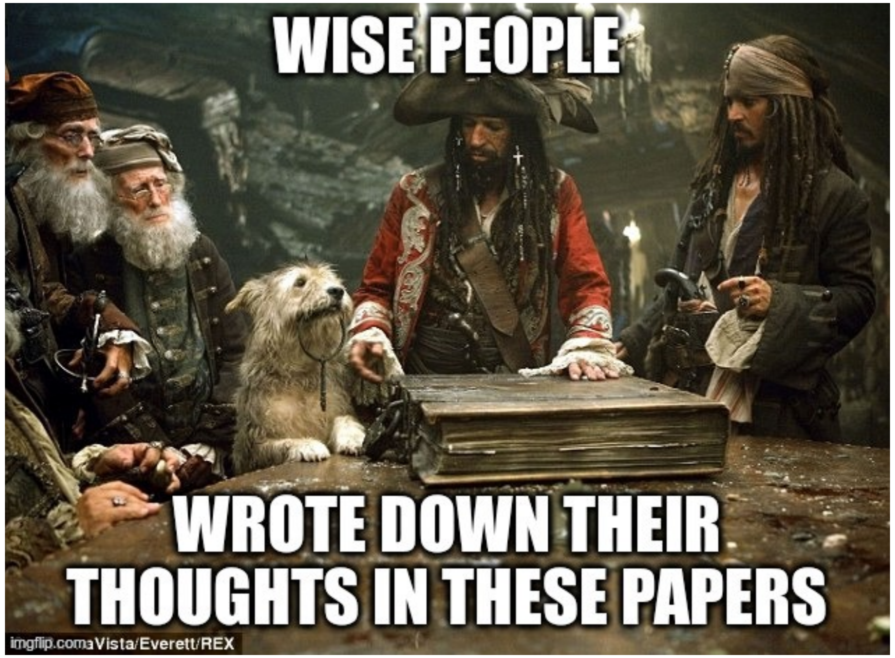
All Beginnings are Difficult

First of all, past publications about multi-arch firmware emulation were studied:

- Automated Dynamic Firmware Analysis at Scale:A Case Study on Embedded Web Interfaces (Costin et al.) – standard Debian Images / **`chroot` in the target firmware**
- FIRMADYNE (Chen et al.) – **modified kernels** with `musl-libc` / target firmware file system is directly used

Both projects were mostly covering ARM(EL), MIPS(BE/EL)

Combined with Buildroot and QEMU



Preparations

The following tasks were crucial to create an environment where the target firmware feels comfortable:

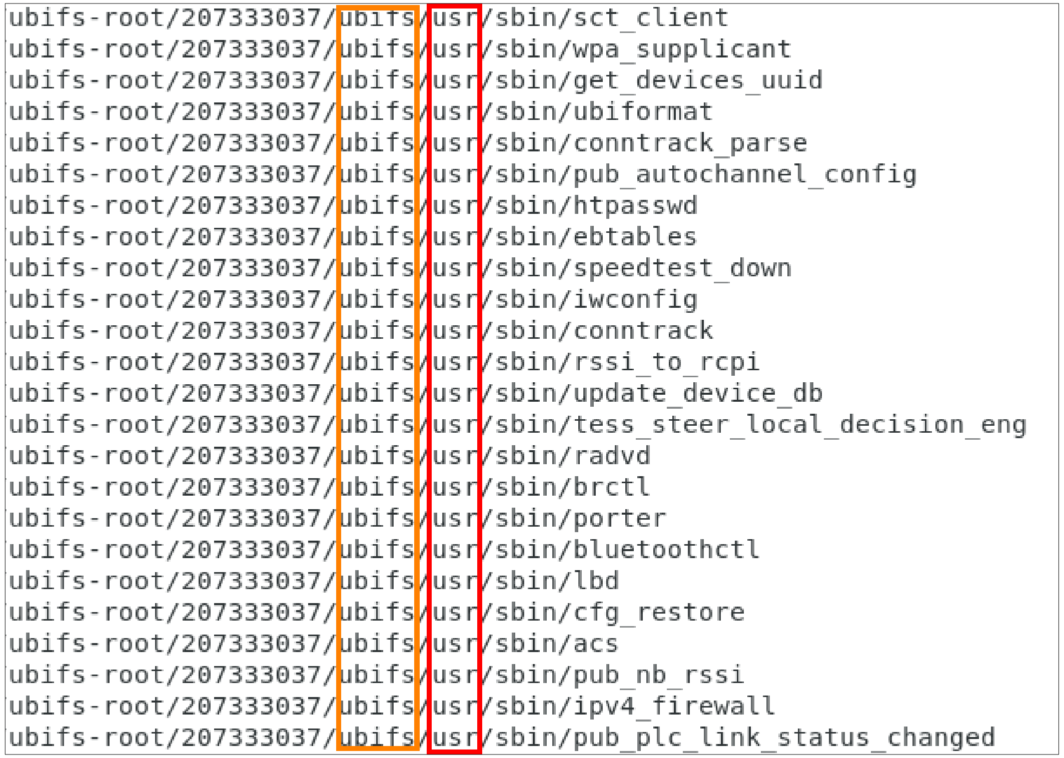
- Find out where the root file system is located → important for the `chroot` command
- Find out what architecture **and** instruction set is used → ARMv4 != ARMv7
- Find out which C library is used → `uClibc`, `musl-libc` or `glibc`
- Prepare a system startup script → `inittab`? `rcS`? Scripts in `rc.d/` or `init.d/` ?

Locate the Root File System

The first thought I had was: “*that must be solved with a complex algorithm*”

But it was much simpler: Do it graphically with a folder keyword search for UNIX based systems.

The only constraint was, that It must be a Linux-based firmware with a file system.



```
ubifs-root/207333037/ubifs/usr/sbin/sct_client
ubifs-root/207333037/ubifs/usr/sbin/wpa_supplicant
ubifs-root/207333037/ubifs/usr/sbin/get_devices_uuid
ubifs-root/207333037/ubifs/usr/sbin/ubiformat
ubifs-root/207333037/ubifs/usr/sbin/conntrack_parse
ubifs-root/207333037/ubifs/usr/sbin/pub_autochannel_config
ubifs-root/207333037/ubifs/usr/sbin/htpasswd
ubifs-root/207333037/ubifs/usr/sbin/ebtables
ubifs-root/207333037/ubifs/usr/sbin/speedtest_down
ubifs-root/207333037/ubifs/usr/sbin/iwconfig
ubifs-root/207333037/ubifs/usr/sbin/conntrack
ubifs-root/207333037/ubifs/usr/sbin/rssi_to_rcpi
ubifs-root/207333037/ubifs/usr/sbin/update_device_db
ubifs-root/207333037/ubifs/usr/sbin/tess_steer_local_decision_eng
ubifs-root/207333037/ubifs/usr/sbin/radvd
ubifs-root/207333037/ubifs/usr/sbin/brctl
ubifs-root/207333037/ubifs/usr/sbin/porter
ubifs-root/207333037/ubifs/usr/sbin/bluetoothctl
ubifs-root/207333037/ubifs/usr/sbin/lbd
ubifs-root/207333037/ubifs/usr/sbin/cfg_restore
ubifs-root/207333037/ubifs/usr/sbin/acs
ubifs-root/207333037/ubifs/usr/sbin/pub_nb_rssi
ubifs-root/207333037/ubifs/usr/sbin/ipv4_firewall
ubifs-root/207333037/ubifs/usr/sbin/pub_plc_link_status_changed
```

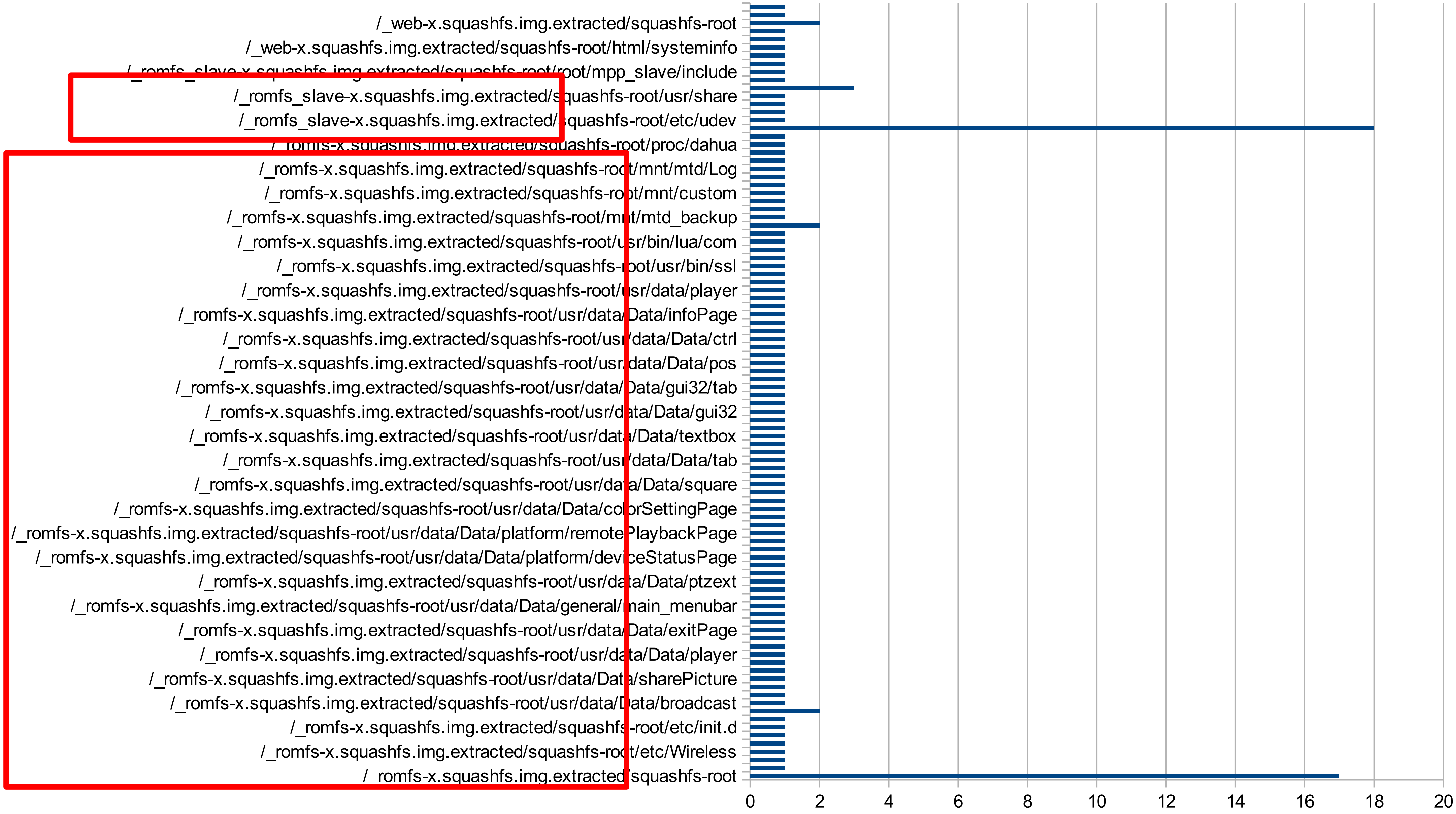
Locate the Root File System

Using this kind of location, a histogram of multiple possible root file systems can also be created.

Do not rely on key binaries like `busybox` or `bash` as they can also be located in a rescue file system.

Use plausibility checks, like “*are there even executables in the detected file system?*”

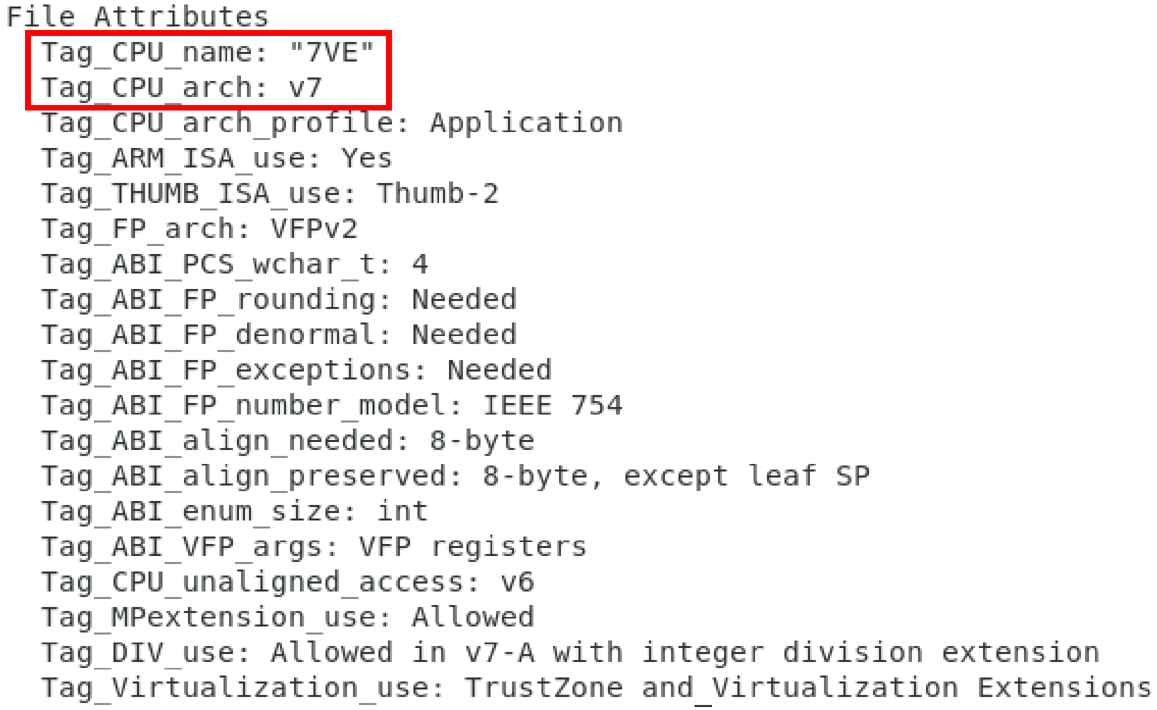
Root File-System Detection



Identify the Architecture

A common way are the tools “readelf” and “file”. But for emulation, a more precise way to identify the exact instruction set is necessary. Other ways are:

- Looking for the “vermagic” string in kernel modules
- Looking for symbols that contain keywords like “ARM7TDMI” or “MIPS32R5”
- If everything fails, grep all executables to find the instruction set (bad success rate)

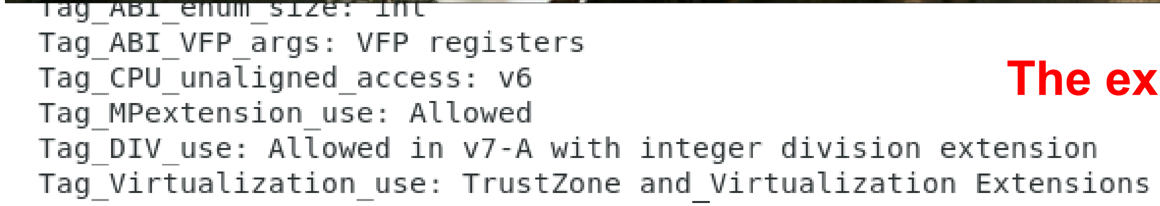


```
File Attributes
  Tag_CPU_name: "7VE"
  Tag_CPU_arch: v7
  Tag_CPU_arch_profile: Application
  Tag_ARM_ISA_use: Yes
  Tag_THUMB_ISA_use: Thumb-2
  Tag_FP_arch: VFPv2
  Tag_ABI_PCS_wchar_t: 4
  Tag_ABI_FP_rounding: Needed
  Tag_ABI_FP_denormal: Needed
  Tag_ABI_FP_exceptions: Needed
  Tag_ABI_FP_number_model: IEEE 754
  Tag_ABI_align_needed: 8-byte
  Tag_ABI_align_preserved: 8-byte, except leaf SP
  Tag_ABI_enum_size: int
  Tag_ABI_VFP_args: VFP registers
  Tag_CPU_unaligned_access: v6
  Tag_MPextension_use: Allowed
  Tag_DIV_use: Allowed in v7-A with integer division extension
  Tag_Virtualization_use: TrustZone and Virtualization Extensions
```

Identify the Architecture

A common way are the tools “readelf” and “file”. But for emulation, a more precise way to identify the exact instruction set is necessary. Other ways are:

- Looking for the “vermagic” string in kernel modules
- Lookin or “MIPS32R5”
- If every et (bad success rate)



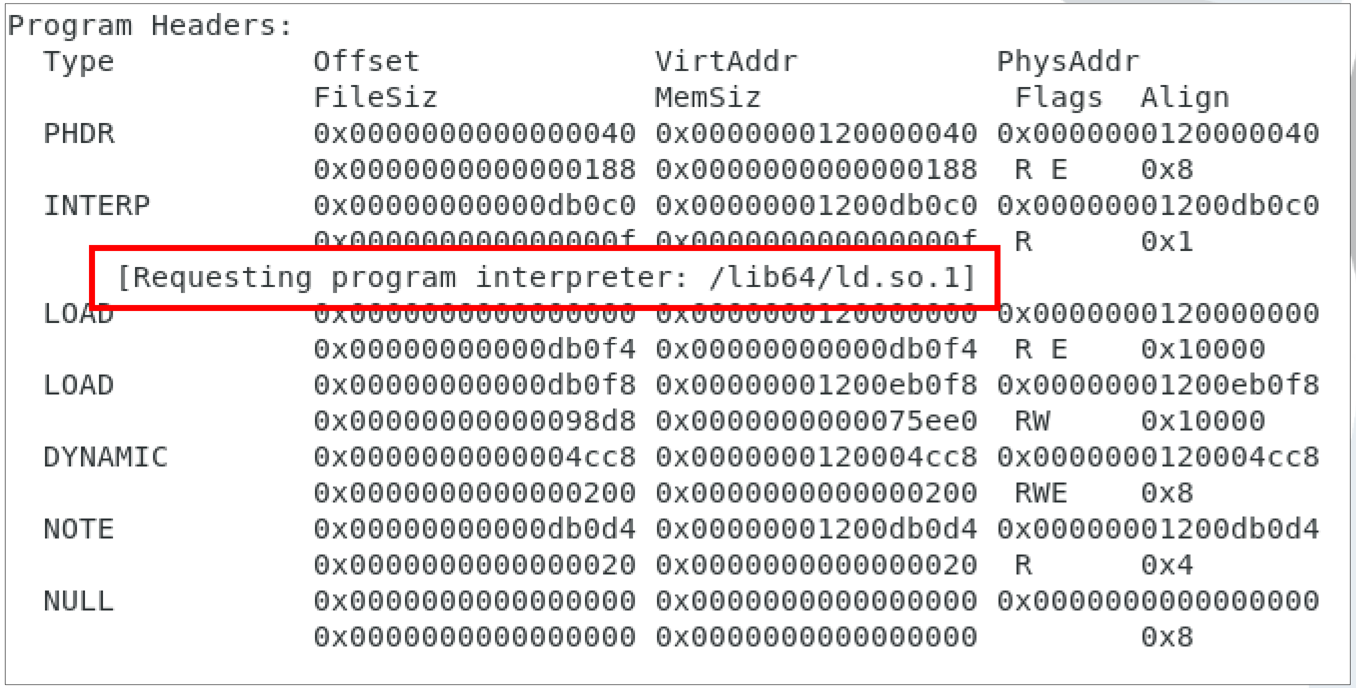
```
Tag_ABI_enum_size: int
Tag_ABI_VFP_args: VFP registers
Tag_CPU_unaligned_access: v6
Tag_MPextension_use: Allowed
Tag_DIV_use: Allowed in v7-A with integer division extension
Tag_Virtualization_use: TrustZone and Virtualization Extensions
```

The exact instruction set matters!

Libraries are Relevant!

The interpreter for executables is especially relevant for cross-compiling binaries.

They can be easily determined and constitute another important detail that must be considered!



```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000120000040 0x0000000120000040
                 0x0000000000000188 0x0000000000000188  R E    0x8
  INTERP         0x00000000000db0c0 0x00000001200db0c0 0x00000001200db0c0
                 0x000000000000000f 0x000000000000000f  R      0x1
      [Requesting program interpreter: /lib64/ld.so.1]
  LOAD           0x0000000000000000 0x0000000120000000 0x0000000120000000
                 0x00000000000db0f4 0x00000000000db0f4  R E    0x10000
  LOAD           0x00000000000db0f8 0x00000001200eb0f8 0x00000001200eb0f8
                 0x00000000000098d8 0x0000000000075ee0  RW     0x10000
  DYNAMIC        0x0000000000004cc8 0x0000000120004cc8 0x0000000120004cc8
                 0x0000000000000200 0x0000000000000200  RWE    0x8
  NOTE           0x00000000000db0d4 0x00000001200db0d4 0x00000001200db0d4
                 0x0000000000000020 0x0000000000000020  R      0x4
  NULL           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000         0x8
```

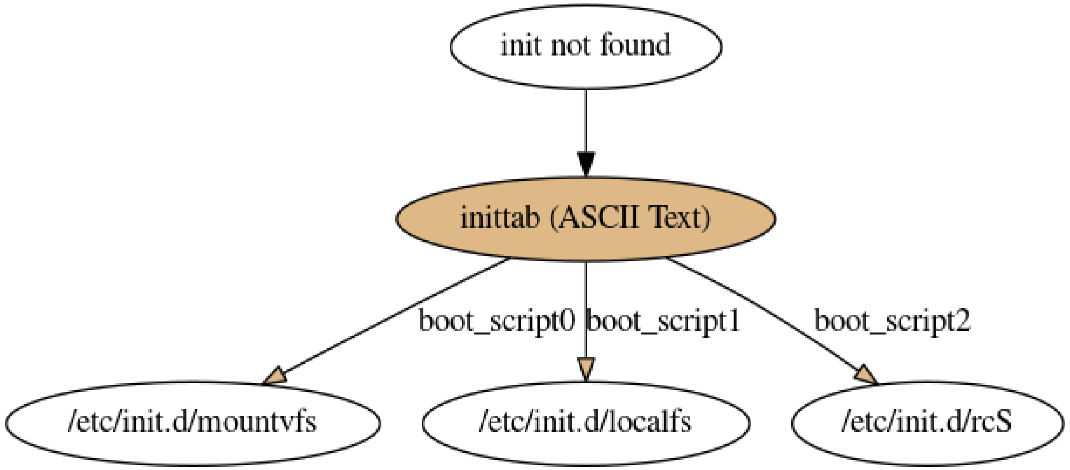
This is crucial for ARM Hard-Float, as it has effect to the QEMU virtual machine.

Script Preparation

Startup scripts in firmwares are placed on different locations on the system.

A straightforward way that was used to start the most firmwares of the sample set is parsing the `inittab` file.

A simple script was written that searches the `inittab` file in the extracted firmware image and also prints a graph of the different commands:



If no `inittab` file is present on the system, other typical startup pointers are `/etc/rcS`, `/etc/init.d/rcS` or `/etc/rc.d/rcS`.

Pre-Analysis – Sample Set of (Almost) 200 Firmwares across 49 Vendors

Interpreters that were found in all the firmwares (sometimes there is more than one):

uClibc	libc (ld-linux/ld)	libc hard-float	musl libc	musl hard-float
103	103 (78/25)	11	3	1

Architectural distribution:

ARMv8 (BE/EL)	ARMv7 (BE/EL)	ARMv6 (BE/EL)	ARMv5 (BE/EL)	ARMv4 (BE/EL)
0 / 2	0 / 58	0 / 3	0 / 34	1 / 5

MIPS32 (BE/EL)	MIPS64 (BE/EL)	PowerPC (BE/EL)	X86 / X86_64	AVR / ARC5
39 / 36	3 / 0	10 / 0	4 / 1	1 / 2

Pre-Analysis – Sample Set of (Almost) 200 Firmwares across 49 Vendors

Interpreters that were found in all the firmwares (sometimes there is more than one):

uClibc	libc (ld-linux/ld)	libc hard-float	musl libc	musl hard-float
103	103 (78/25)	11	3	1

Architectural distribution:

ARM is very often used as little endian architecture, but it differs for MIPS!

ARMv8 (BE/EL)	ARMv7 (BE/EL)	ARMv6 (BE/EL)	ARMv5 (BE/EL)	ARMv4 (BE/EL)
0 / 2	0 / 58	0 / 3	0 / 34	1 / 5
MIPS32 (BE/EL)	**MIPS64 (BE/EL)**	**PowerPC (BE/EL)**	**X86 / X86_64**	**AVR / ARC5**
39 / 36	3 / 0	10 / 0	4 / 1	1 / 2

Pre-Analysis – Sample Set of (Almost) 200 Firmwares across 49 Vendors

Interpreters that were found in all the firmwares (sometimes there is more than one):

uClibc	libc (ld-linux/ld)	libc hard-float	musl libc	musl hard-float
103	103 (78/25)	11	3	1

Architectural distribution:

ARMv8 (BE/EL)	ARMv7 (BE/EL)	ARMv6 (BE/EL)	ARMv5 (BE/EL)	ARMv4 (BE/EL)
0 / 2	0 / 58	0 / 3	0 / 34	1 / 5

MIPS32 (BE/EL)	MIPS64 (BE/EL)	PowerPC (BE/EL)	X86 / X86_64	AVR / ARC5
39 / 36	3 / 0	10 / 0	4 / 1	1 / 2

Cavium Octeon: Documentation available@
http://vsevteme.ru/network/169/attachments/show?content=297548
http://vsevteme.ru/network/169/attachments/show?content=297550

Very specific architectures
E.g. SH4 is used for other industries

Preparing Fake Images – Buildroot to the Rescue!

Our workflow:

Done!

1) Pre-analysis of the target firmware

2) Creating a suitable firmware image with kernel and userland (for analysis)

3) Copy the identified root file-system into the created firmware image

4) Start the firmware image and use `chroot` to switch into the target firrmware

5) Run all startup scripts

6) Security analysis

Preparing Fake Images – Buildroot to the Rescue!

Our workflow:

1) Pre-analysis of the target firmware

No problem with Buildroot

2) Creating a suitable firmware image with kernel and userland (for analysis)

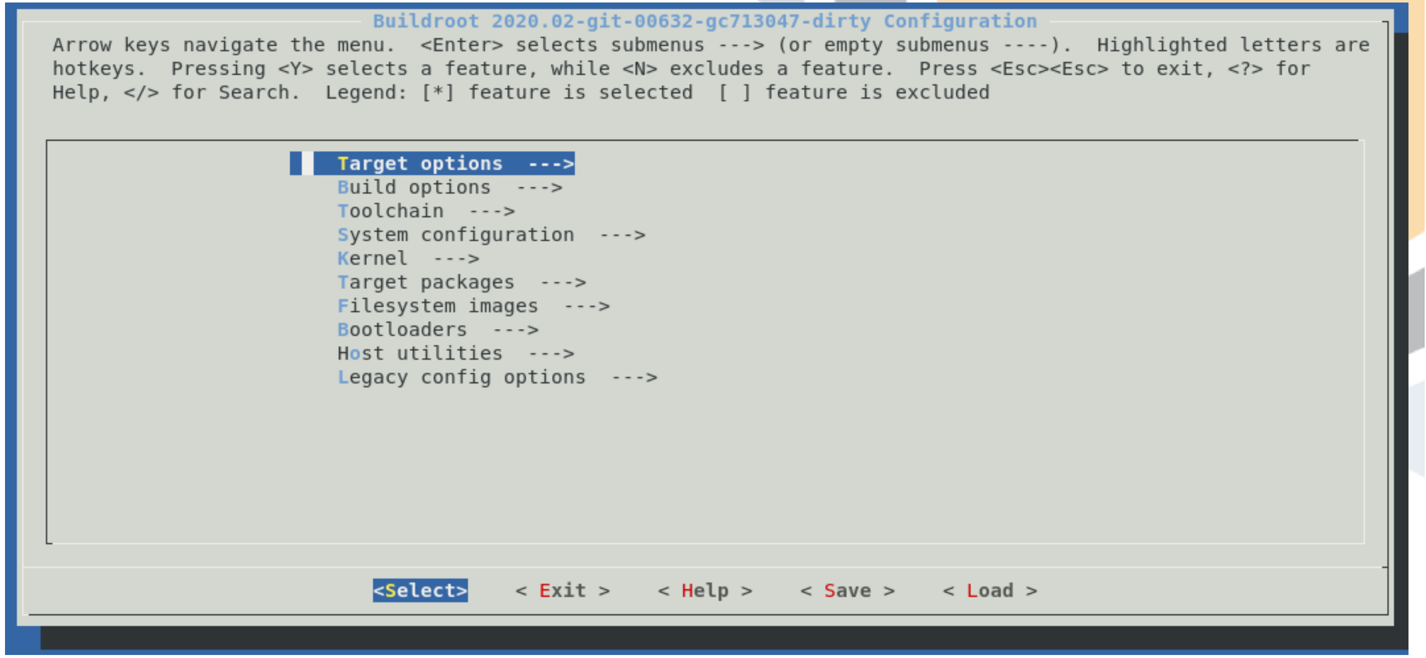
3) Copy the identified root file-system into the created firmware image

4) Start the firmware image and use `chroot` to switch into the target firrmware

5) Run all startup scripts

6) Security analysis

A cross-compiler can also be generated!



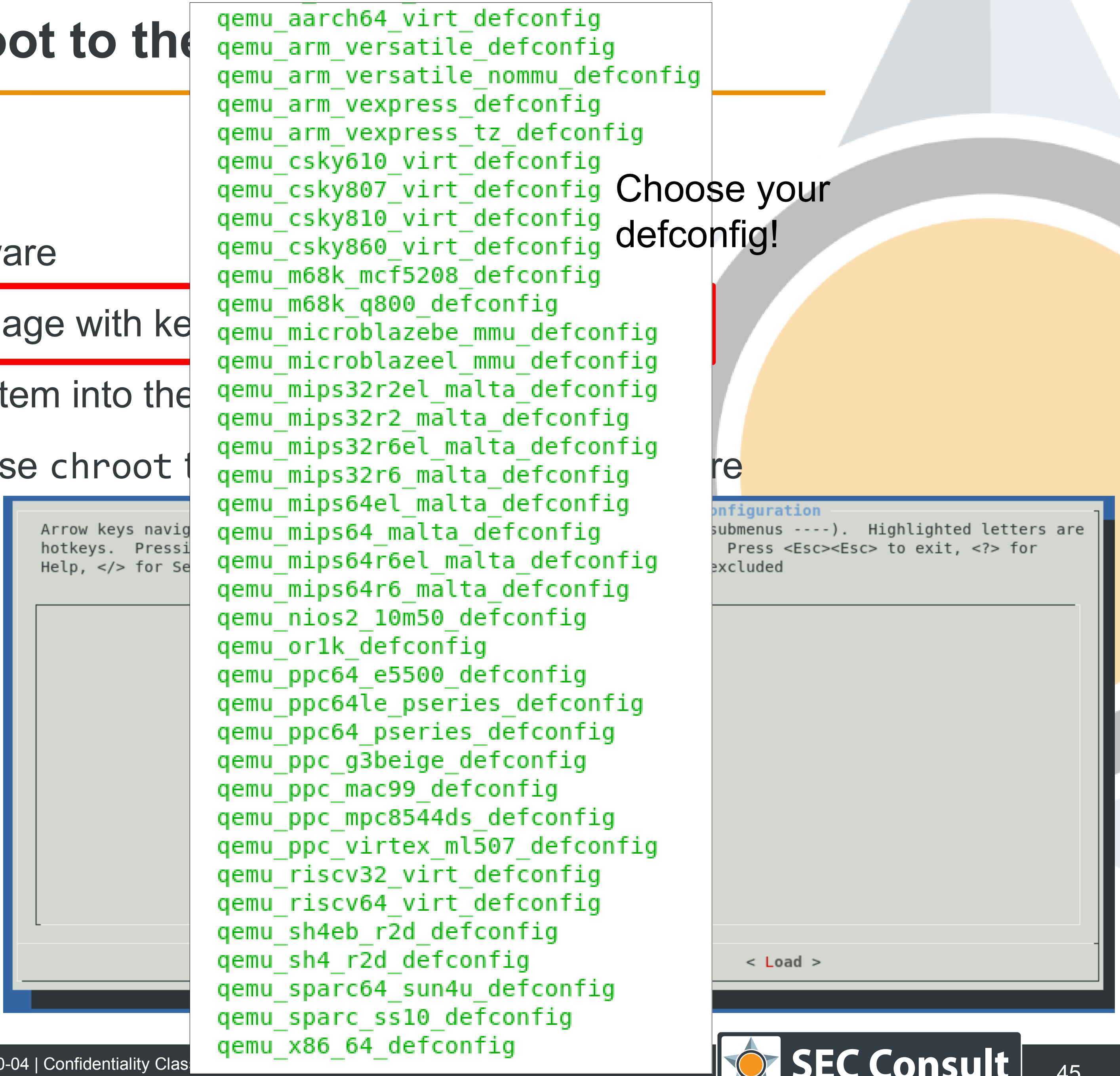
```
Buildroot 2020.02-git-00632-gc713047-dirty Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are
hotkeys.  Pressing <Y> selects a feature, while <N> excludes a feature.  Press <Esc><Esc> to exit, <?> for
Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is excluded
```

Preparing Fake Images – Buildroot to the

Our workflow:

1) Pre-analysis of the target firmware
2) Creating a suitable firmware image with ke
3) Copy the identified root file-system into the
4) Start the firmware image and use `chroot` t re
5) Run all startup scripts
6) Security analysis

A cross-compiler can also be generated!



```
qemu_aarch64_virt_defconfig
qemu_arm_versatile_defconfig
qemu_arm_versatile_nommu_defconfig
qemu_arm_vexpress_defconfig
qemu_arm_vexpress_tz_defconfig
qemu_csky610_virt_defconfig
qemu_csky807_virt_defconfig
qemu_csky810_virt_defconfig
qemu_csky860_virt_defconfig
qemu_m68k_mcf5208_defconfig
qemu_m68k_q800_defconfig
qemu_microblazebe_mmu_defconfig
qemu_microblazeel_mmu_defconfig
qemu_mips32r2el_malta_defconfig
qemu_mips32r2_malta_defconfig
qemu_mips32r6el_malta_defconfig
qemu_mips32r6_malta_defconfig
qemu_mips64el_malta_defconfig
qemu_mips64_malta_defconfig
qemu_mips64r6el_malta_defconfig
qemu_mips64r6_malta_defconfig
qemu_nios2_10m50_defconfig
qemu_or1k_defconfig
qemu_ppc64_e5500_defconfig
qemu_ppc64le_pseries_defconfig
qemu_ppc64_pseries_defconfig
qemu_ppc_g3beige_defconfig
qemu_ppc_mac99_defconfig
qemu_ppc_mpc8544ds_defconfig
qemu_ppc_virtex_ml507_defconfig
qemu_riscv32_virt_defconfig
qemu_riscv64_virt_defconfig
qemu_sh4eb_r2d_defconfig
qemu_sh4_r2d_defconfig
qemu_sparc64_sun4u_defconfig
qemu_sparc_ss10_defconfig
qemu_x86_64_defconfig
```

```
                                                     onfiguration
Arrow keys navig                                     submenus ----).  Highlighted letters are
hotkeys.  Pressi                                       Press <Esc><Esc> to exit, <?> for
Help, </> for Se                                     excluded
```

Choose your defconfig!

Preparing Fake Images – Buildroot to the Rescue!

Our workflow:

1) Pre-analysis of the target firmware

2) Creating a suitable firmware image with kernel and userland (for analysis)

Straightforward

3) Copy the identified root file-system into the created firmware image

4) Start the firmware image and use `chroot` to switch into the target firrmware

5) Run all startup scripts

6) Security analysis

Preparing Fake Images – Buildroot to the Rescue!

Our workflow:

1) Pre-analysis of the target firmware

2) Creating a suitable firmware image with kernel and userland (for analysis)

3) Copy the identified root file-system into the created firmware image

4) Start the firmware image and use `chroot` to switch into the target firrmware

5) Run all startup scripts

6) Security analysis

Does not work for all cases.
Sometimes just partial images are available!

Preparing Fake Images – Buildroot to the Rescue!

Our workflow:

1) Pre-analysis of the target firmware

2) Creating a suitable firmware image with kernel and userland (for analysis)

3) Copy the identified root file-system into the created firmware image

4) Start the firmware image and use `chroot` to switch into the target firrmware

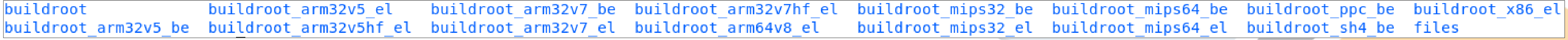
5) Run all startup scripts

6) Security analysis

Does not work for all cases.
Sometimes just partial images are available!

Preparing Fake Images – Covered Architectures

A lot of architectures were covered for this project.



```
buildroot              buildroot_arm32v5_el    buildroot_arm32v7_be  buildroot_arm32v7hf_el  buildroot_mips32_be  buildroot_mips64_be  buildroot_ppc_be  buildroot_x86_el
buildroot_arm32v5_be   buildroot_arm32v5hf_el  buildroot_arm32v7_el  buildroot_arm64v8_el    buildroot_mips32_el  buildroot_mips64_el  buildroot_sh4_be  files
```

The target firmware is embedded into the firmware image to keep the network traffic low. Mounting via NFS was the first way how it was done, but that turned out to be not optimal for monitoring and debugging.

A bridged network was used within QEMU to start the firmware with a dedicated PC.

By changing the hardware address for each firmware with QEMU command line parameters and using DHCP, the full process can be designed scalable.

Loading kernel modules is only possible when the version is fitting!

Firmware Emulation Demo – Runtime!

© www.awn.com/news/pirates-caribbean-reboot-rises-davey-jones-locker

Monitoring and Debugging

To watch the firmware startup and called commands and network daemons, monitoring and debugging is an important step. Findings for an easier life:

- Most important are the Linux commands `ps`, `top`, `netstat` and `tcpdump`.
- During the evaluation, a good portion of valuable information was gathered just by dumping the output of `netstat` and `ps`.
- The painful cross-compilation can be skipped as Buildroot covers this :)
- Static builds of `gdb(-server)`, `strace`, `ltrace`, `valgrind` and other tools can be done with this toolchain.

The kernel can also be customized with Buildroot:

- enable tracing and use `perf` to get all calls!
- this can be done with $ `make linux-menuconfig`

Study Samples from ...

Nexans
BRINGS ENERGY TO LIFE

mercury systems

SAUCE

perle

Actiontec

COMTROL
Pepperl+Fuchs

PHOENIX CONTACT

DrayTek

QNAP

KATHREIN

belkin

BOSCH

D-Link
Building Networks for People

RED LION
a spectris company

WAGO

MOXA

ASUS

aruba
a Hewlett Packard Enterprise company

tp-link

ZyXEL

devolo

AZTECH

LINKSYS

EZVIZ

Parrot

ADVANTECH

Honeywell

Tenda

AXIS COMMUNICATIONS

Beijer ELECTRONICS

ASRock

RITTAL

westermo

LANTRONIX

imperial

ALLNET

EDIMAX
NETWORKING PEOPLE TOGETHER

UBIQUITI NETWORKS

SIEMENS

PLANET
Networking & Communication

WD

korenix
A Beijer Electronics Group Company

NATIONAL INSTRUMENTS

xerox

TRENDnet

Milesight

Schneider Electric

Study Outcome of Linux Based Firmware Emulation

Emulation success rate (tested with `sh/ash/bash`) 178/199 (~89%)

One or more inferred TCP listeners 31/199 (~16%)

One or more inferred UDP listeners 15/199 (~8%)

A lot firmware images were incomplete which is the reason why many services could not have been started. This does not mean that the emulation itself wasn't successful!

Known vulnerabilities that were tested automatically:

- CVE-2015-7547 (glibc getaddinfo buffer overflow) 8/199 (~4%)
- CVE-2015-0235 (GHOST buffer overflow) 28/199 (~14%)
- Shellshock (multiple CVEs) 1/199 (~0.5%)

Selected Vulnerabilities – More Demos

Published:

- **Command Injection in Phoenix Contact Devices**
- **Hardcoded Credentials & Vulnerable TPS in Cisco SMB Routers**
- **Hardcoded Key Material & Vulnerable TPS in WAGO Managed Industrial Switches**

Unpublished but already communicated within our responsible disclosure process to the vendor:

- Multiple Vulnerabilities in one Red Lion Device
- Multiple Vulnerabilities in Korenix Devices

More vulnerabilities that must be communicated...

Command Injection in Phoenix Contact Devices – Analysis

By loading the “cfg” CGI binary into Ghidra, the vulnerable code can be spotted very fast:

```
    uVar4 = scan_boundary(0,*(undefined4 *)(param_2 + 8),3,0,0);
    return uVar4;
  }
  if (param_3 != 0) {
    return 0;
  }
  if (*(int *)(param_1 + 0x10) != 1) goto LAB_000093e8;
  if (*(char **)(param_1 + 0x14) == (char *)0x0) {
    html_printf(1,"<pre>%s</pre>\r\n","missing filename");
    goto LAB_000093e8;
  }
  __sl = strrchr(*(char **)(param_1 + 0x14),0x2e);
  if (__sl == (char *)0x0) {
    uVar4 = *(undefined4 *)(param_1 + 0xc);
LAB_0000948c:
    __sl = "";
  }
  else {
    __sl = __sl + 1;
    uVar4 = *(undefined4 *)(param_1 + 0xc);
    if (__sl == (char *)0x0) goto LAB_0000948c;
  }
  run_shell(0x1000,"/usr/sbin/import_cfg /tmp/cfg_import %s/new_config %s",uVar4,__sl);
  html_printf(1,"<pre>%s</pre>\r\n","please reboot next");
LAB_000093e8:
  print_foot();
  free(*(void **)(param_1 + 0x14));
  remove_dir_leaf(*(undefined4 *)(param_1 + 0xc));
  return 0xffffffff;
}
```

Conclusion and Further Work

Emulating firmware with QEMU and Buildroot while covering different architectures works really good!

Tested approaches were:

- Pre-built Debian images → no kernel modifications possible, changes in the userland are hard.
- Building the kernel from scratch → kernel modifications are really complex, only good when you are familiar with Linux kernel internals.
- Using the target firmware’s file system only → observation must be done via QEMU and the kernel, manual testing is hard.

Improvements:

- Implement Cavium Octeon to QEMU (KVM already supports this architecture)
- Use kernel hopping in Buildroot → enables loading of some kernel modules
- Library resolving, e.g. by using `scanelf` → helps to reconstruct the file system
- Pre-emulation with QEMU user mode → better architecture detection

Thank You!