Firmware reverse engineering And emulation

2020/8/31

- Shengchikuo
- cipher@onwardsecurity.com
- Shengchikuo@gmail.com

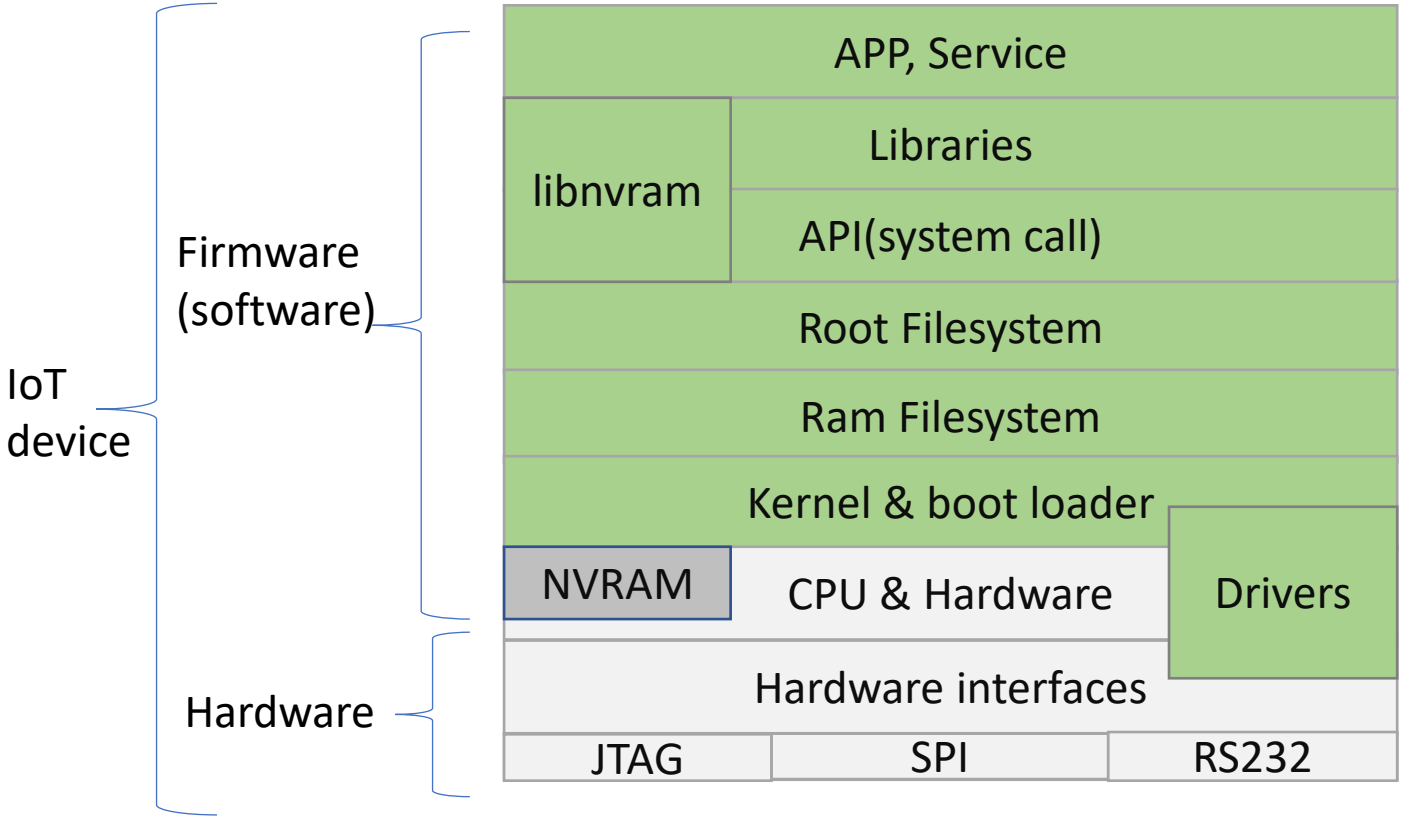
ONWARD SECURITY
SECURE YOUR OWN WORLD !

Introduction

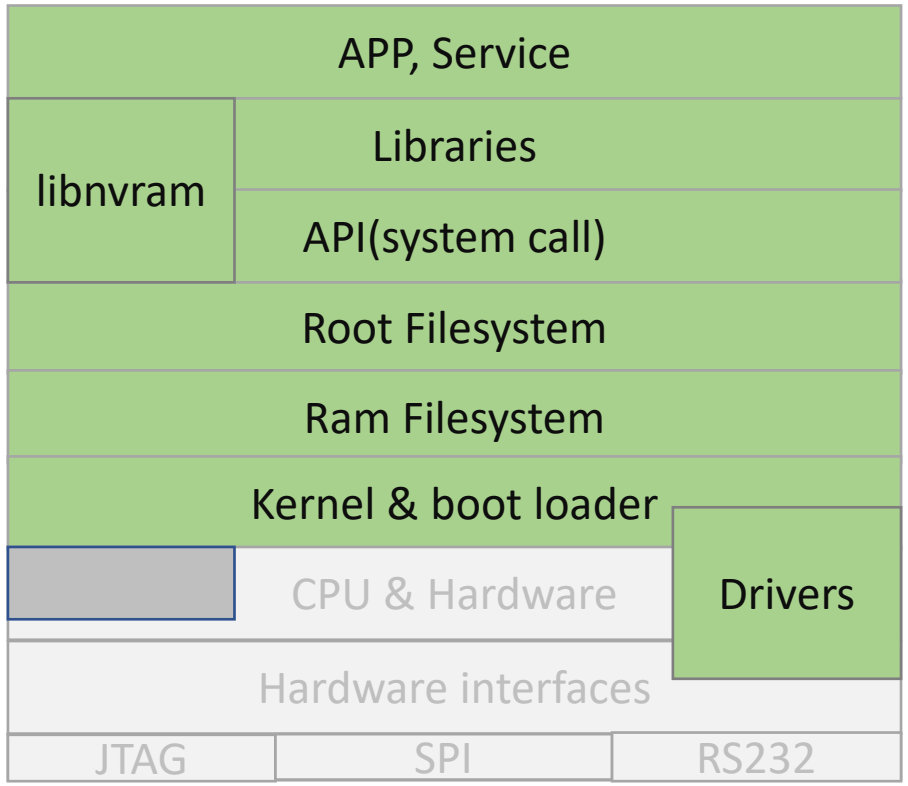
- This talk is related to the firmware reverse engineering, So I think it would better go through the basic prerequisite knowledge first.

What Is firmware?

- Firmware is a tar ball of OS, custom-drivers, application software that make the IoT device work, just like your desktop and laptop.
- firmware is a specific class of computer software that provides the low-level control for a device's specific hardware.
- And it is roughly composited with these components.(figure in right)



How does embedded system engineers manufacture one IoT device(software part)



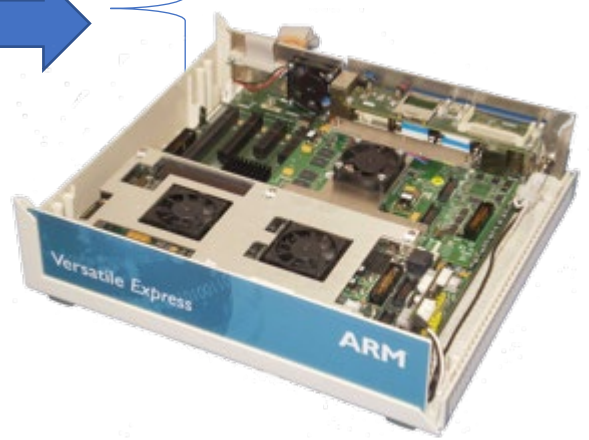
firmware

1. Create loop device.
2. mount virtual storage.
3. make eMMC standard partition.
4. Make file systems.
5. Put kernel, driver, application, etc. to the specific position.
6. combine(Compress) everything into one file.

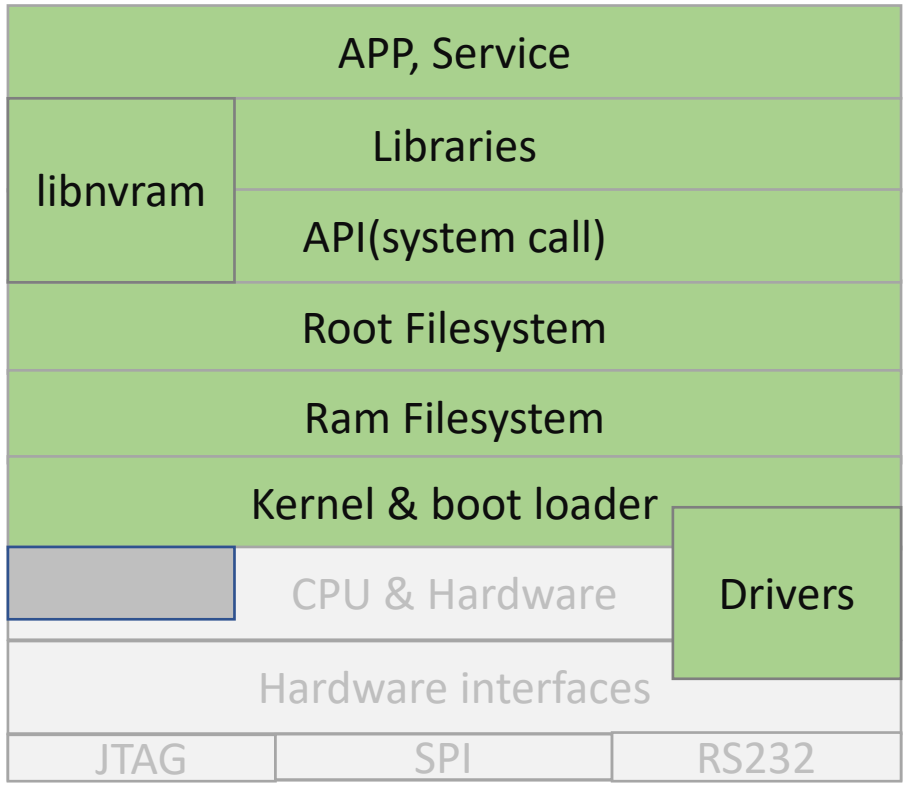
One compressed Binary file

Fire into

ROM,
EPROM
flash memory
Etc.



And this is how do we Reverse engineering this stuff.



firmware

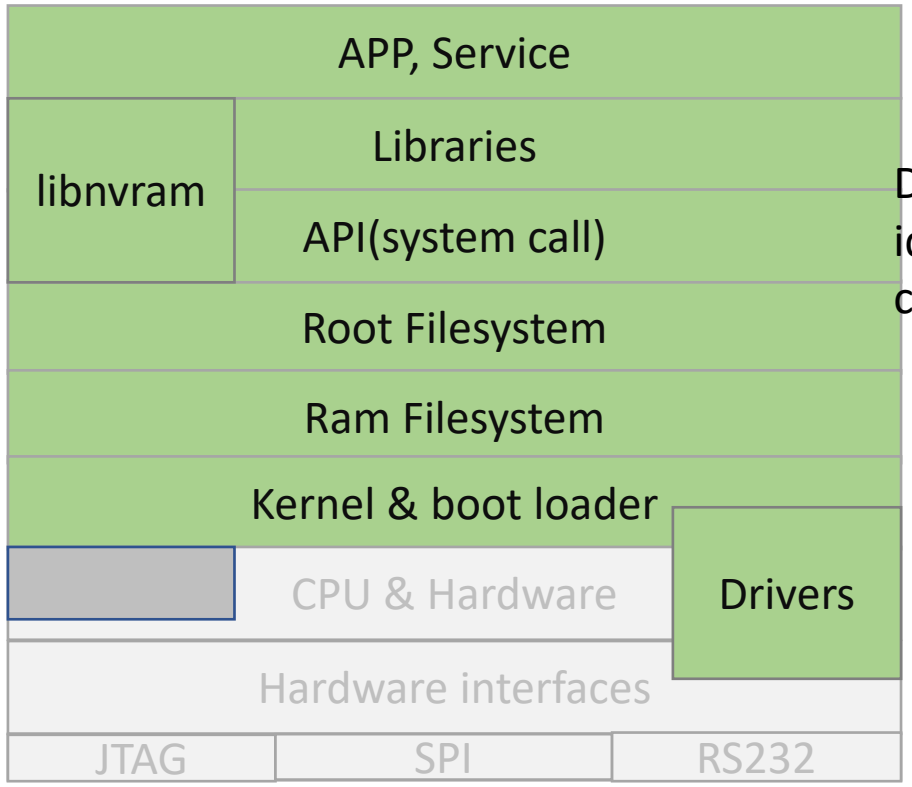
Decompress it and identify each component

One compressed Binary file

Dump the binary data From storage with JTAG

ROM, EPROM flash memory Etc.

What Is firmware? And how to Reverse engineering this stuff.



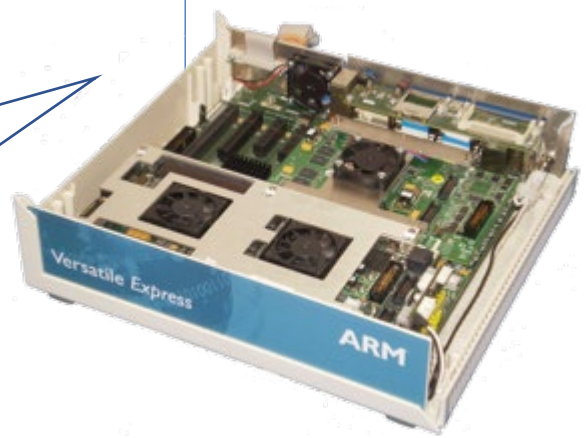
firmware

Decompress it and identify each component

One compressed Binary file

Dump the binary data with JTAG

ROM, EPROM flash memory Etc.



But we don't always have enough money to buy one expensive device, or it may have read protection on the device.

What Is firmware? And how to Reverse engineering this stuff.

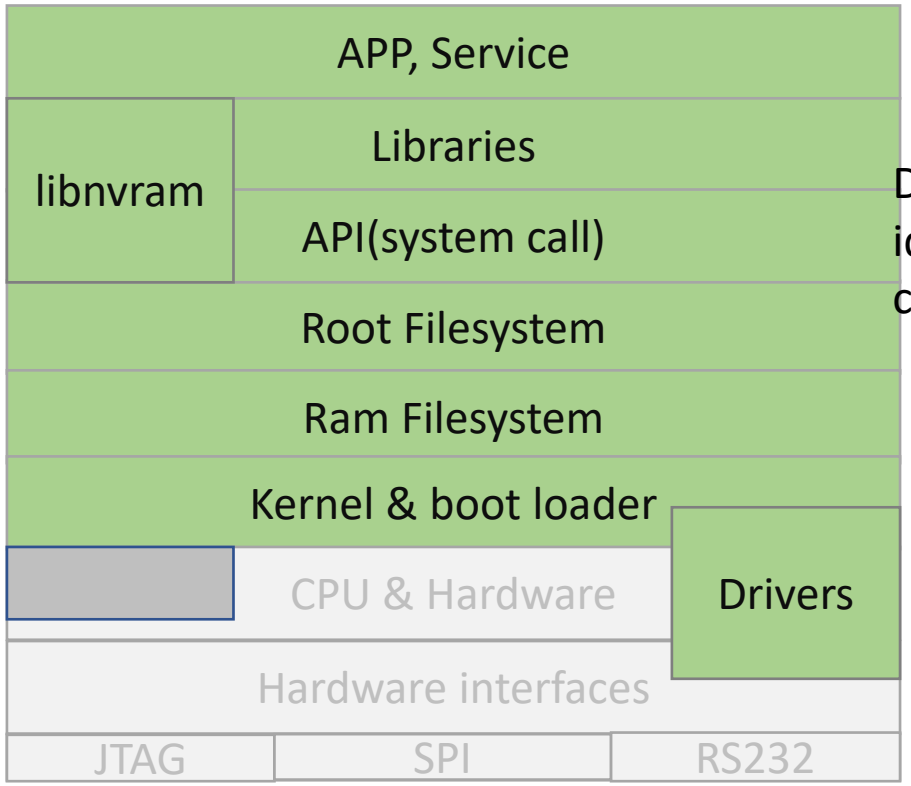
Download from official website Of IoT device manufacture.

Dump the binary data with JTAG

ROM, EPROM flash memory Etc.

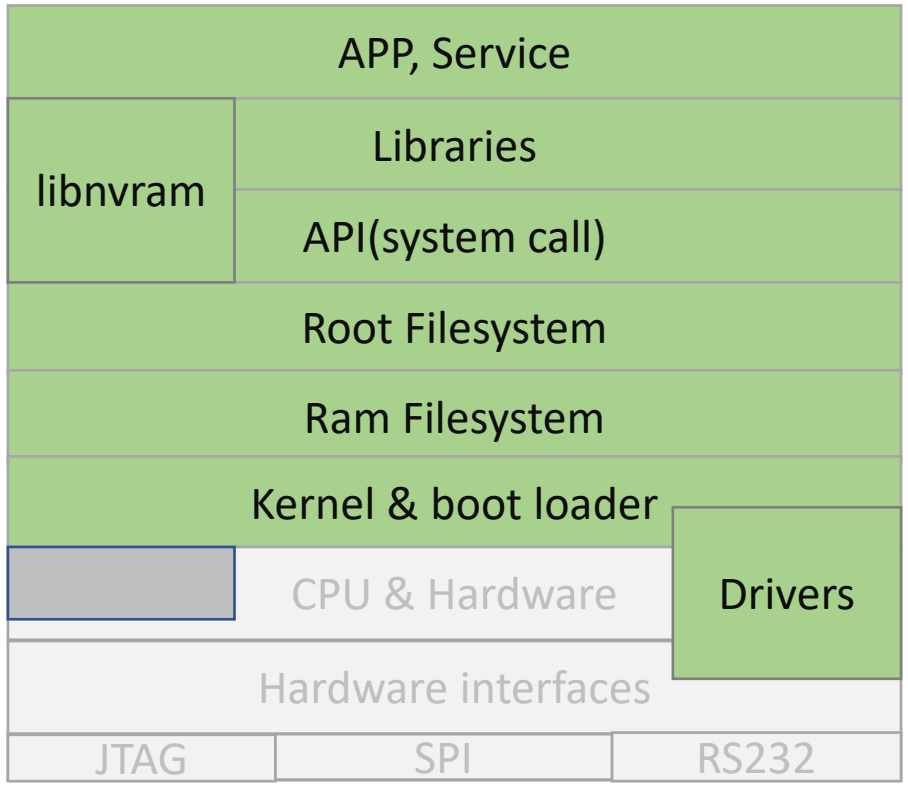
One compressed Binary file

Decompress it and identify each component



firmware

What Is firmware? And how to Reverse engineering this stuff.



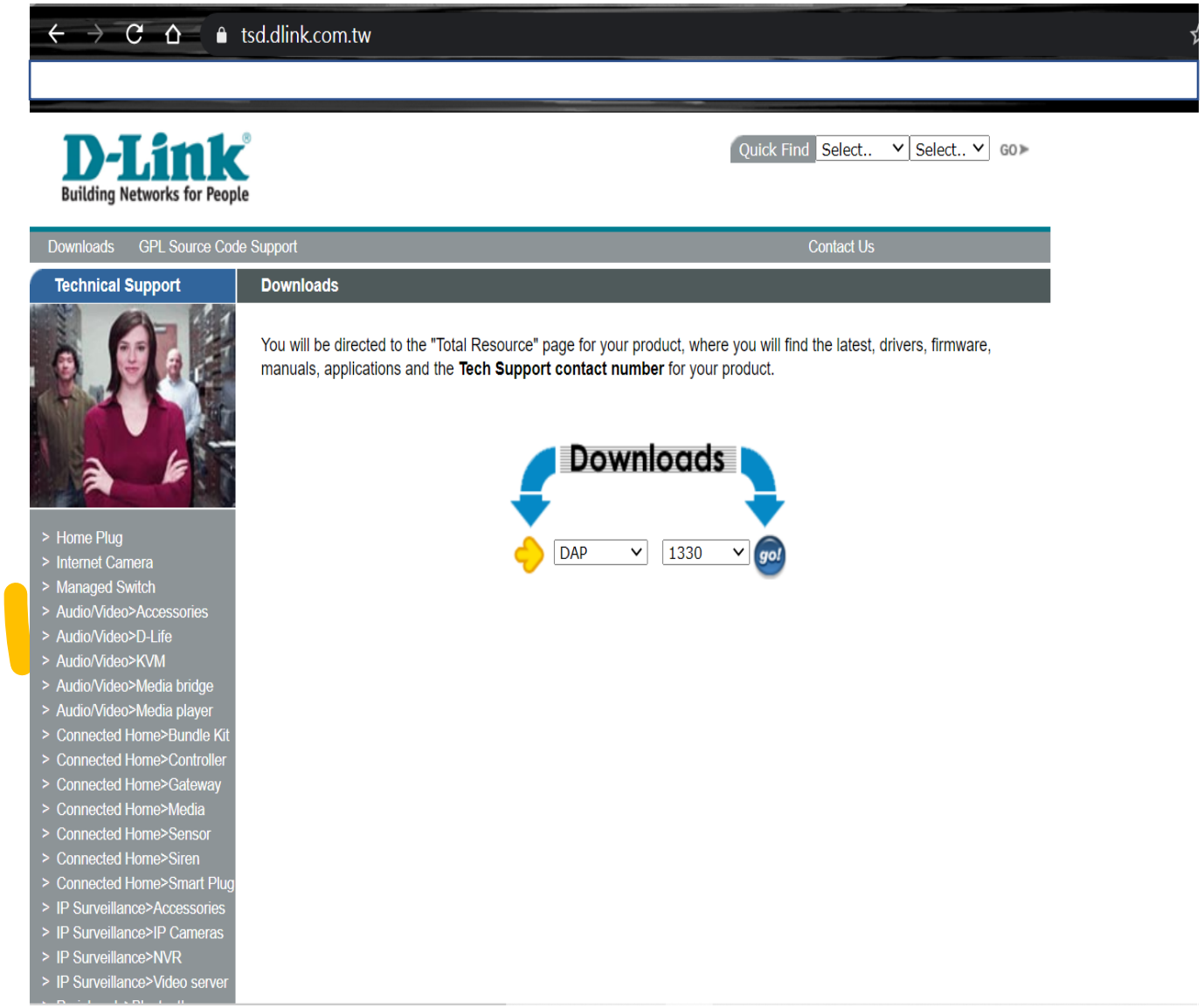
firmware

- Decompress
- identify each component

One compressed Binary file

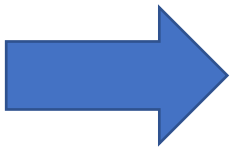
Or you can sniff the network trace between the device and server from manufacturer, when the firmware update.

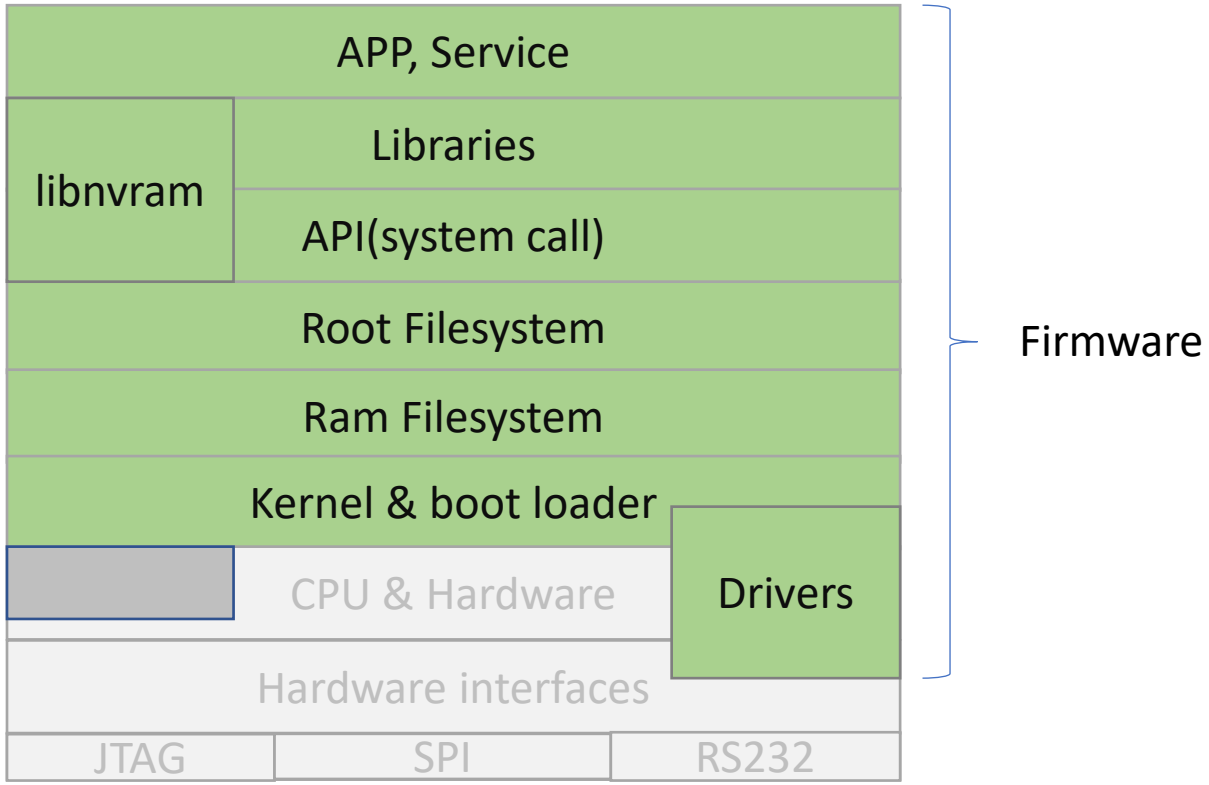
Get your firmware from manufacturer



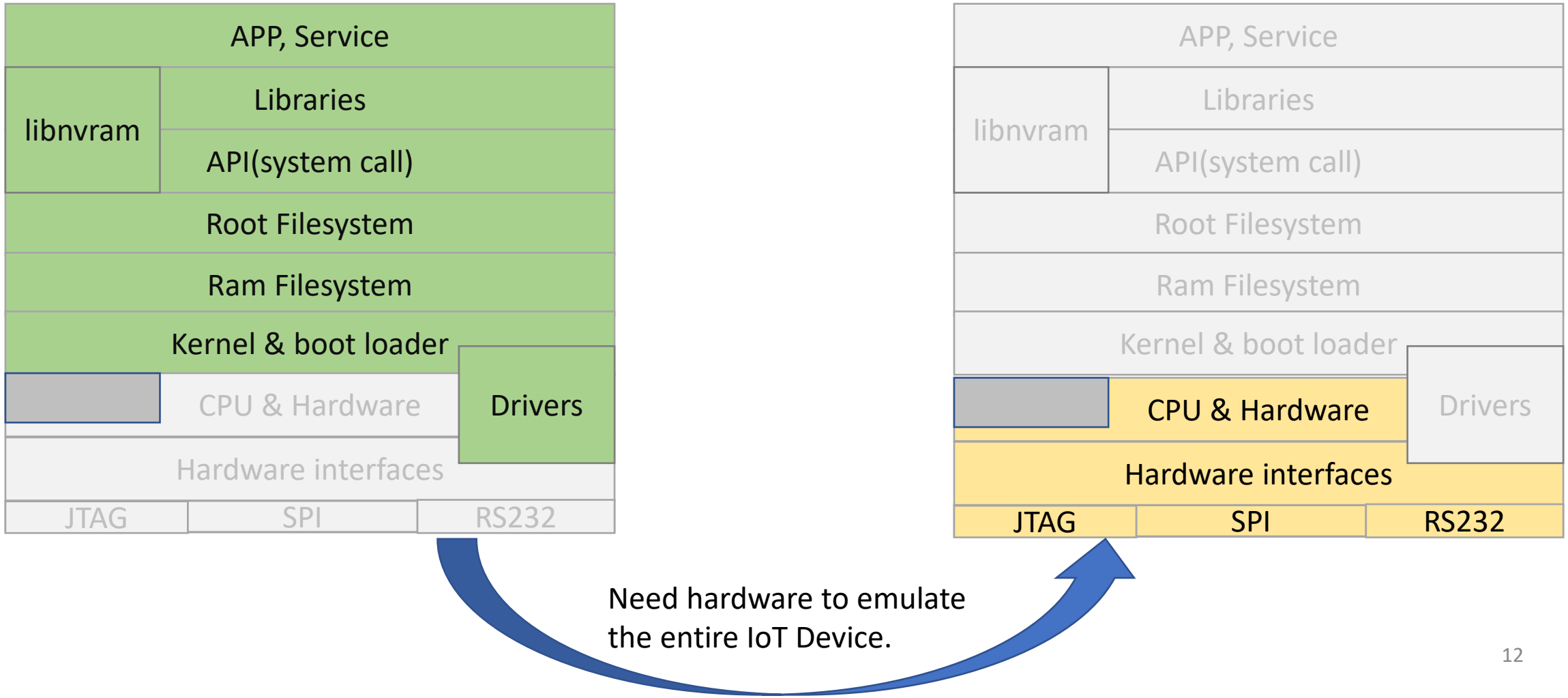
- Here is an instance, how you canjust download the firmware from D-Link.
- If you want to follow the practice latter, it would better to know where to get one firmware first.

So We got the software from previous Step

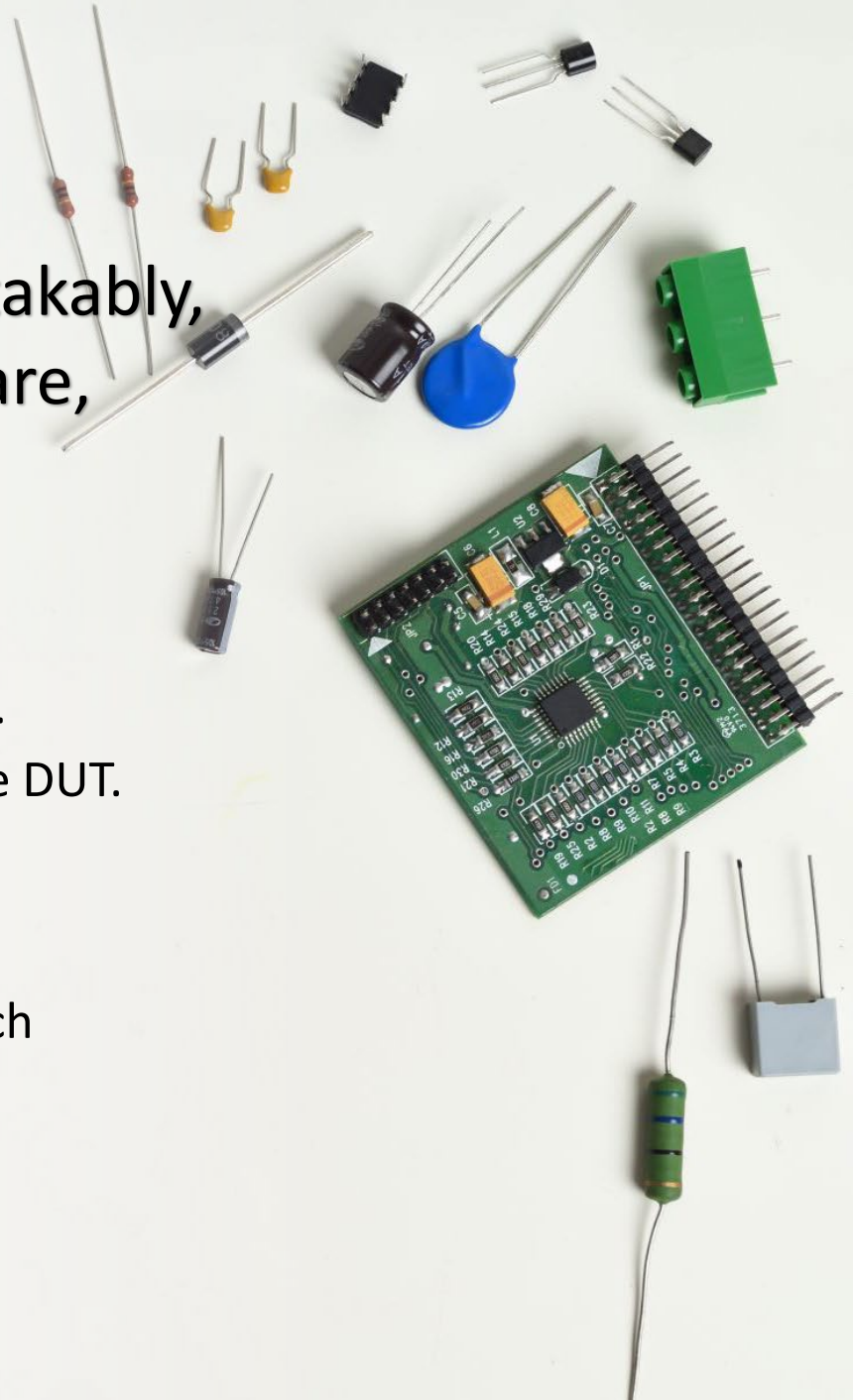




Now we need filled missing part (hardware) to emulate everything unmistakably



Now we need hardware to settle everything unmistakably, That means we need both hardware and software, but HOW?



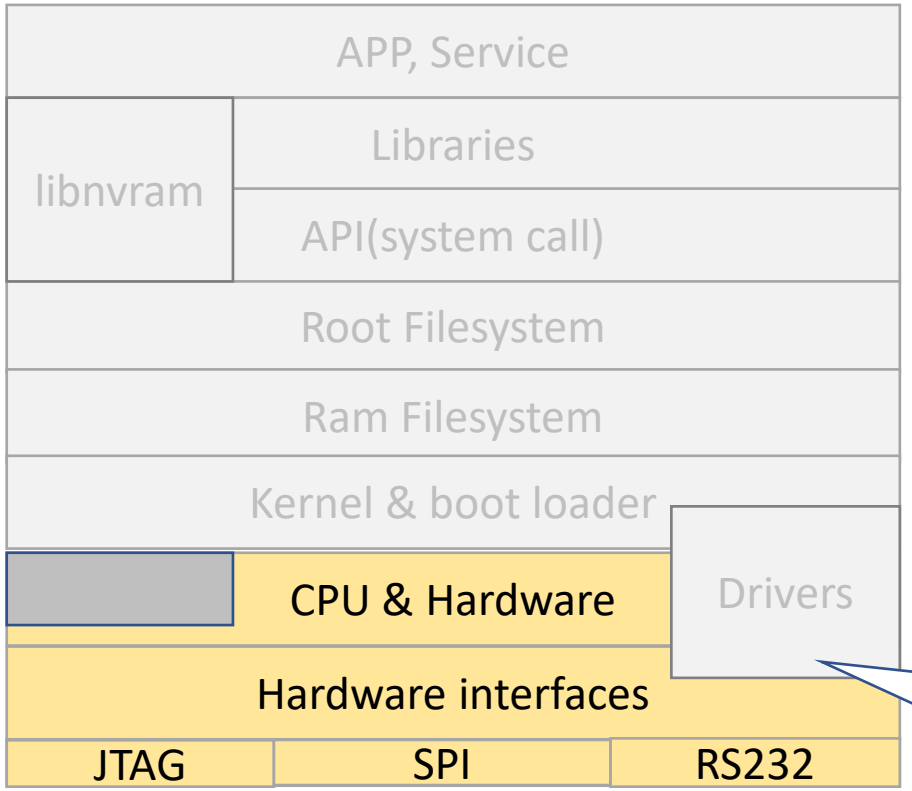
Real device

- We can purchase a development board with same model as the DUT.
- Fire back the modified firmware which content debugging tool to the DUT.

Virtual machine

How about just emulating the hardware part with virtual machine which enable us save more money and also can fit with diverse architecture?

Utilize the QEMU to emulate hardware

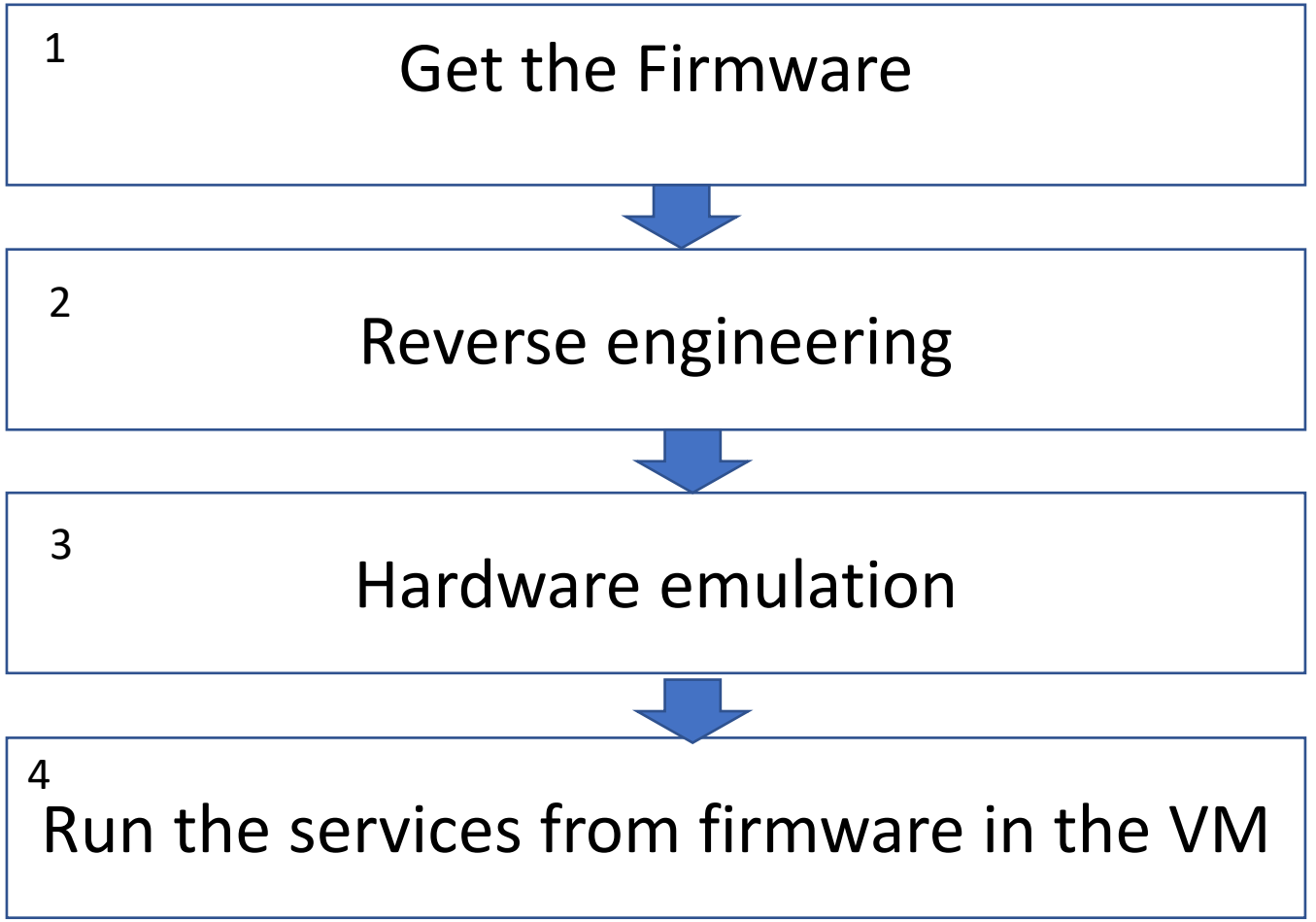


QEMU (short for Quick Emulator)is a free and open-source emulator that performs hardware virtualization.

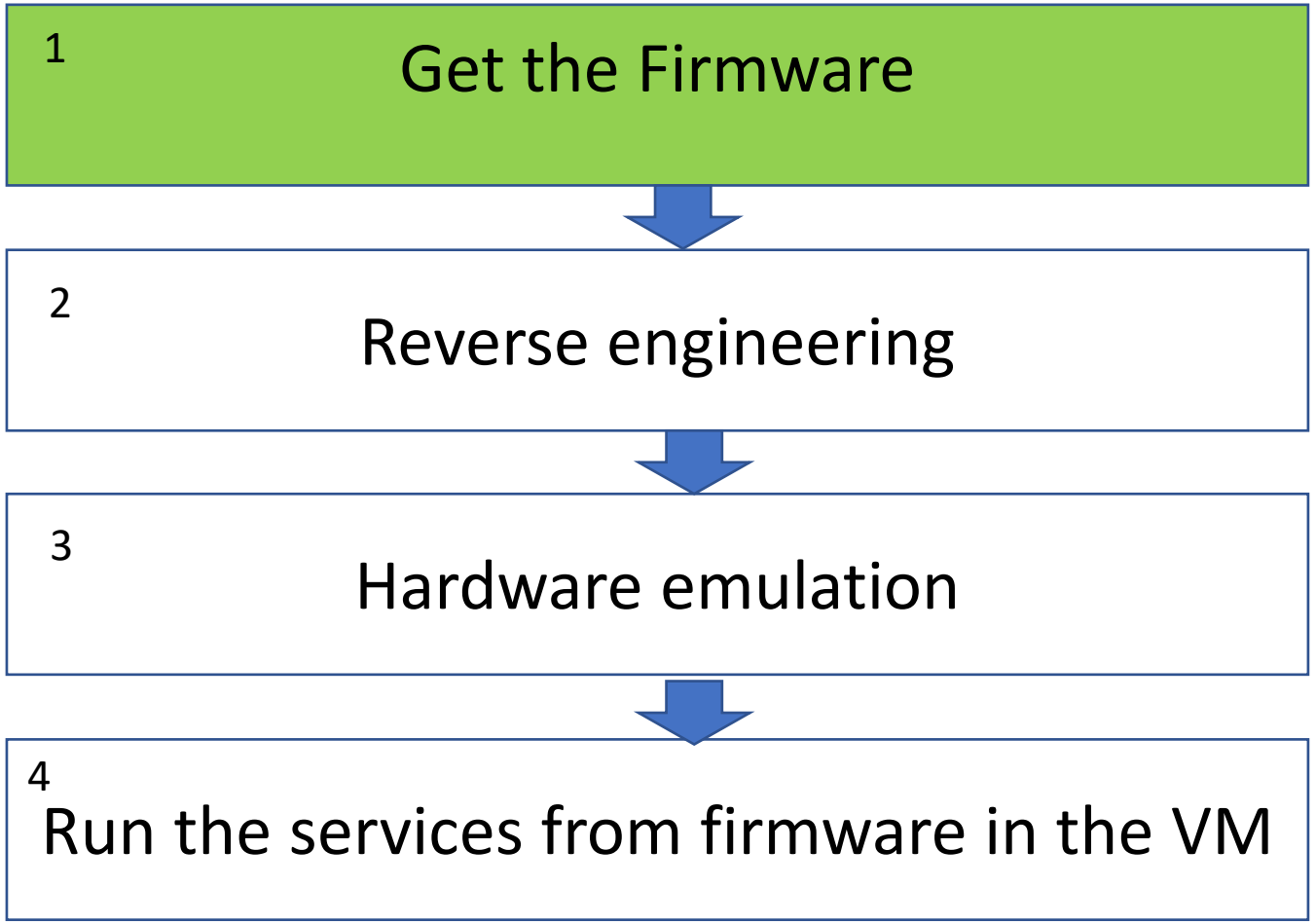
QEMU is a hosted virtual machine monitor: it emulates the machine's processor through dynamic binary translation and provides a set of different hardware and device models for the machine, enabling it to run a variety of guest operating systems.

There is an open-source project QEMU, committed their selves in hardware emulation

we can divide the entire emulation Progress into 4 part.



And, we have gone though the first part.



- But before we go through deeper of the remain parts, maybe it is a good idea to discuss how can we benefit from doing this.
- After all, we still do not talk about how can we benefit from the firmware reverse engineering and emulation.

Consider how can you benefit form it.

Zero-day exploring by fuzzing or other dynamic analysis technique with the firmware running on VM.

Modify the firmware and add some functionality, debugging the original firmware

If you are a software security company, you might want your dynamic analysis service in cloud, so your customer can just upload the firmware, and have a test online.

Apply fuzzing on the VM

Someone may have a question like why just carry out the fuzz test on the device itself, why do we need to fuzzing test the firmware inside the VM.

Think about this.

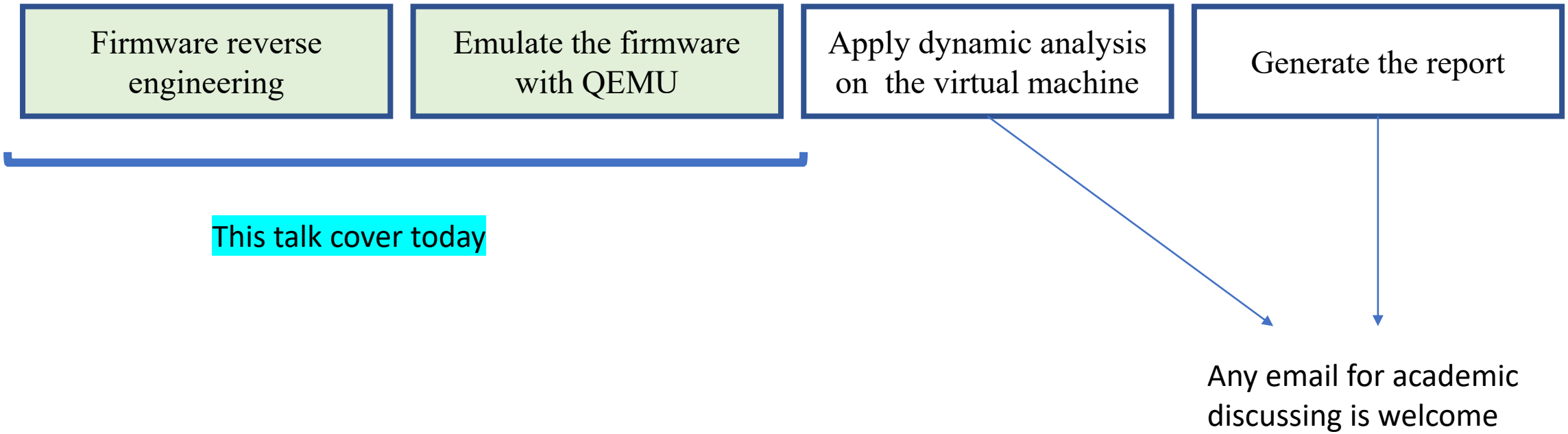
- We can parallelize the original fuzzing test method, to reduce the total fuzzing time.
- Reduce the cost spent on experimental DUT.
- We can dynamic analysis the firmware (Put GDB server inside firmware)

So, here is our long-term goal

Firmware reverse engineering	Emulate the firmware with QEMU	Dynamic analysis the virtual machine	Generate the report

And automatic the process.

The things we are going to cover
In this talk today, due to time limitation

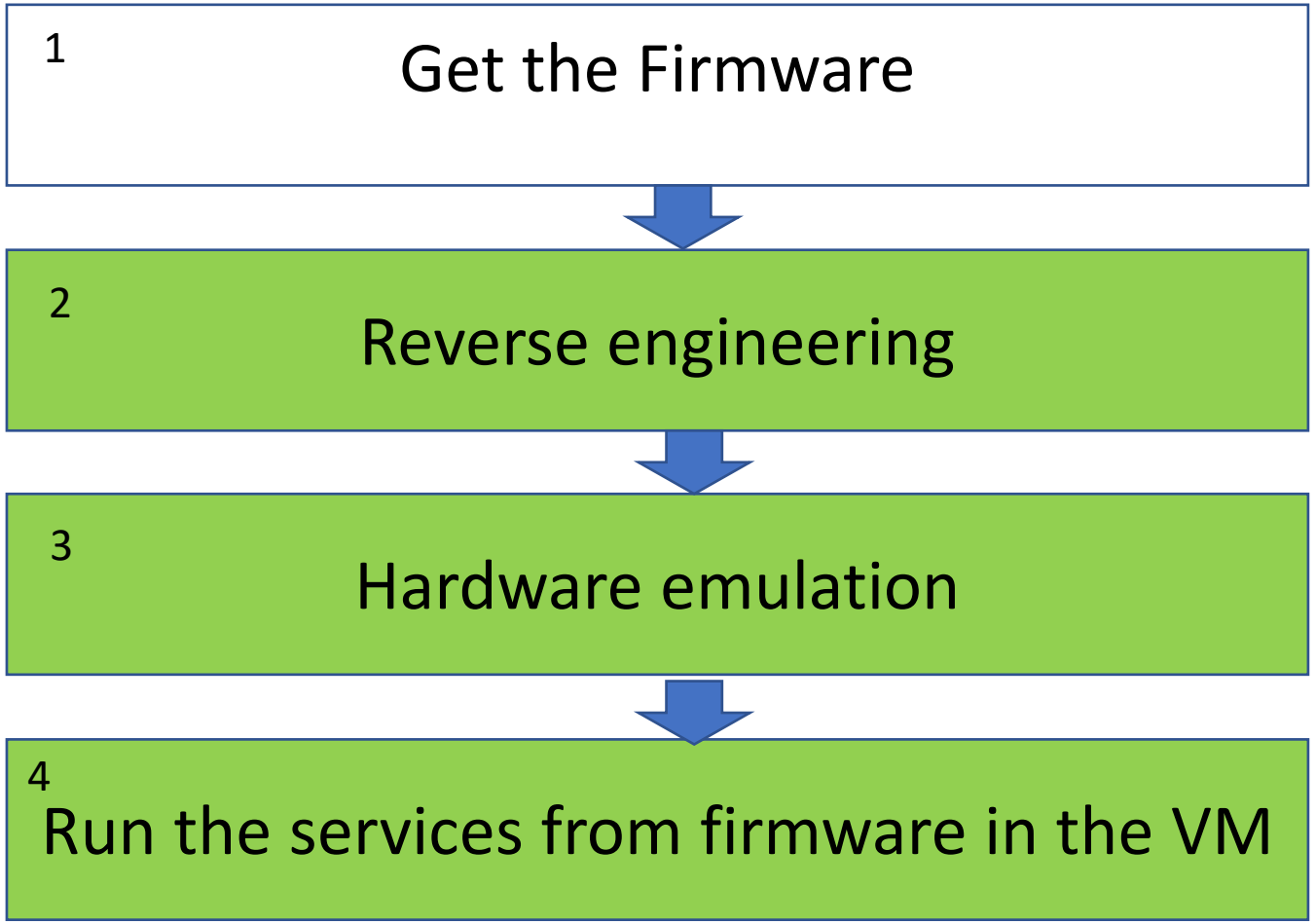


The following talk will be organized as following

- Experiment setup with open source project.
- Talk about limitation and challenges we discovered.
- How to conquer the limitation.
- The problems still need to be solved in the future to achieve the goal.

Experiment setup
With open source project

- Instead of doing the remain parts manually, we can finish them by using open source projects.
- And the things will be handled by them automatically.
- From reverse engineering to run the service in the VM.



Use opensource tool.

Firmadyne

Firmware analysis toolkit

FIRMADYNE:
is an automated and scalable system for performing emulation and dynamic analysis of Linux-based embedded firmware.

Github Link:
https://github.com/firmadyne/firmadyne

Firmware analysis toolkit:
FAT is a toolkit base on Firmadyne, enable the Firmadyne can be automatic in using.

Github Link:
https://github.com/attify/firmware-analysis-toolkit

Installation & environment setup:

```
$git clone firmware-analysis-toolkit
```

```
$cd firmware-analysis-toolkit
```

```
$git clone Firmadyne –recursive
```

```
$cd Firmadyne
```

Install the dependencies post on the GitHub homepage of Firmadyne.

```
$sudo ./setup.sh
```

```
$sudo ./download.sh
```

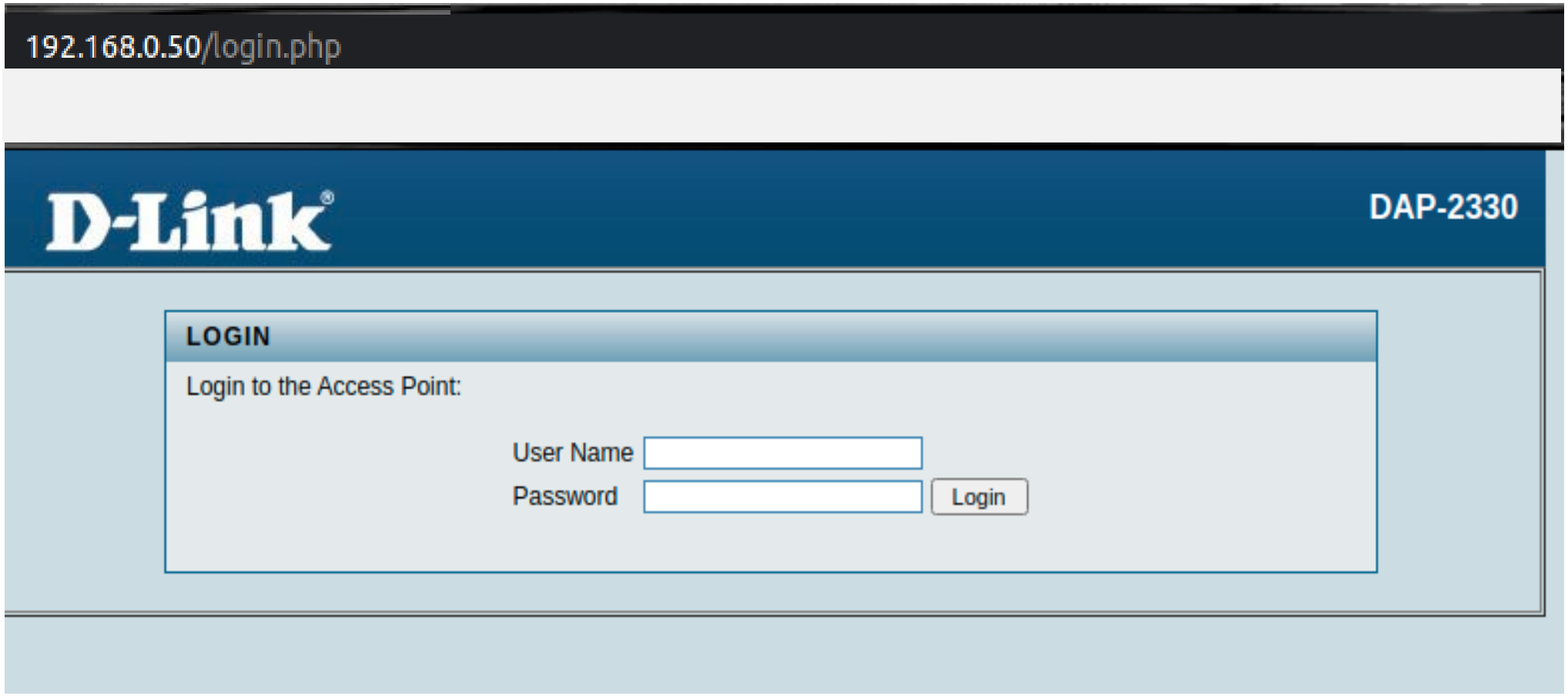
Edit configs

If everything is settled correctly, You can use the following command to emulate the firmware.

```
$sudo python3 fat.py DAP2330-firmware-v110-rc034.bin
```

```
Creating TAP device tap1_0...
Set 'tap1_0' persistent and owned by uid 0
Bringing up TAP device...
Adding route to 192.168.0.50...
Starting firmware emulation... use Ctrl-a + x to exit
[    0.000000] Linux version 2.6.32.70 (vagrant@vagrant-ubuntu-trusty-64) (gcc version 5.3.0 (GCC) ) #1 Thu Feb 18 01:39:21 UTC 2016
[    0.000000]
[    0.000000] LINUX started...
[    0.000000] bootconsole [early0] enabled
[    0.000000] CPU revision is: 00019300 (MIPS 24Kc)
[    0.000000] FPU revision is: 00739300
[    0.000000] Determined physical RAM map:
[    0.000000]  memory: 00001000 @ 00000000 (reserved)
[    0.000000]  memory: 000ef000 @ 00001000 (ROM data)
[    0.000000]  memory: 0061e000 @ 000f0000 (reserved)
[    0.000000]  memory: 0f8f1000 @ 0070e000 (usable)
[    0.000000] debug: ignoring loglevel setting.
[    0.000000] Wasting 57792 bytes for tracking 1806 unused pages
[    0.000000] Initrd not found or empty - disabling initrd
[    0.000000] Zone PFN ranges:
[    0.000000]   DMA      0x00000000 -> 0x00001000
[    0.000000]   Normal   0x00001000 -> 0x0000ffff
[    0.000000] Movable zone start PFN for each node
[    0.000000] early_node_map[1] active PFN ranges
[    0.000000]     0: 0x00000000 -> 0x0000ffff
[    0.000000] On node 0 totalpages: 65535
[    0.000000] free_area_init_node: node 0, pgdat 806aa3c0, node_mem_map 81000000
[    0.000000]   DMA zone: 32 pages used for memmap
[    0.000000]   DMA zone: 0 pages reserved
[    0.000000]   DMA zone: 4064 pages, LIFO batch:0
[    0.000000]   Normal zone: 480 pages used for memmap
[    0.000000]   Normal zone: 60959 pages, LIFO batch:15
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 65023
[    0.000000] Kernel command line: root=/dev/sda1 console=ttyS0 nandsim.parts=64,64,64,64,64,64,64,64,64,64 rdinit=/firmadyne/preInit.sh rw debug
adyne.syscall=0
[    0.000000] PID hash table entries: 1024 (order: 0, 4096 bytes)
[    0.000000] Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)
[    0.000000] Primary instruction cache 2kB, VIPT, 2-way, linesize 16 bytes.
```

later, you can access the service run on the current computer, via the virtual adapter, here is the network interface of the firmware.

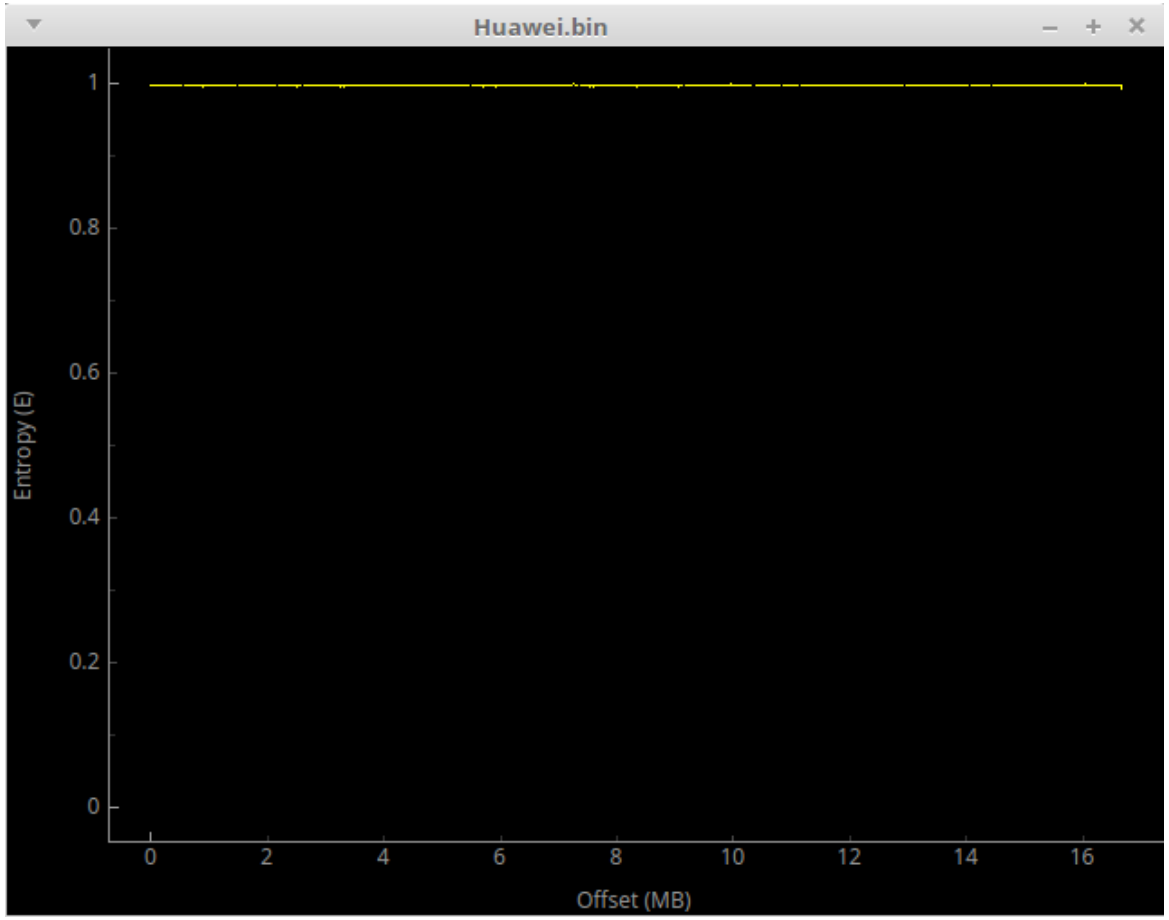


- By the way, before you conduct the firmware emulation with the toolkit I have mentioned, it is important to make sure the firmware didn’t be encrypted, You can use entropy to have a preliminarily check.

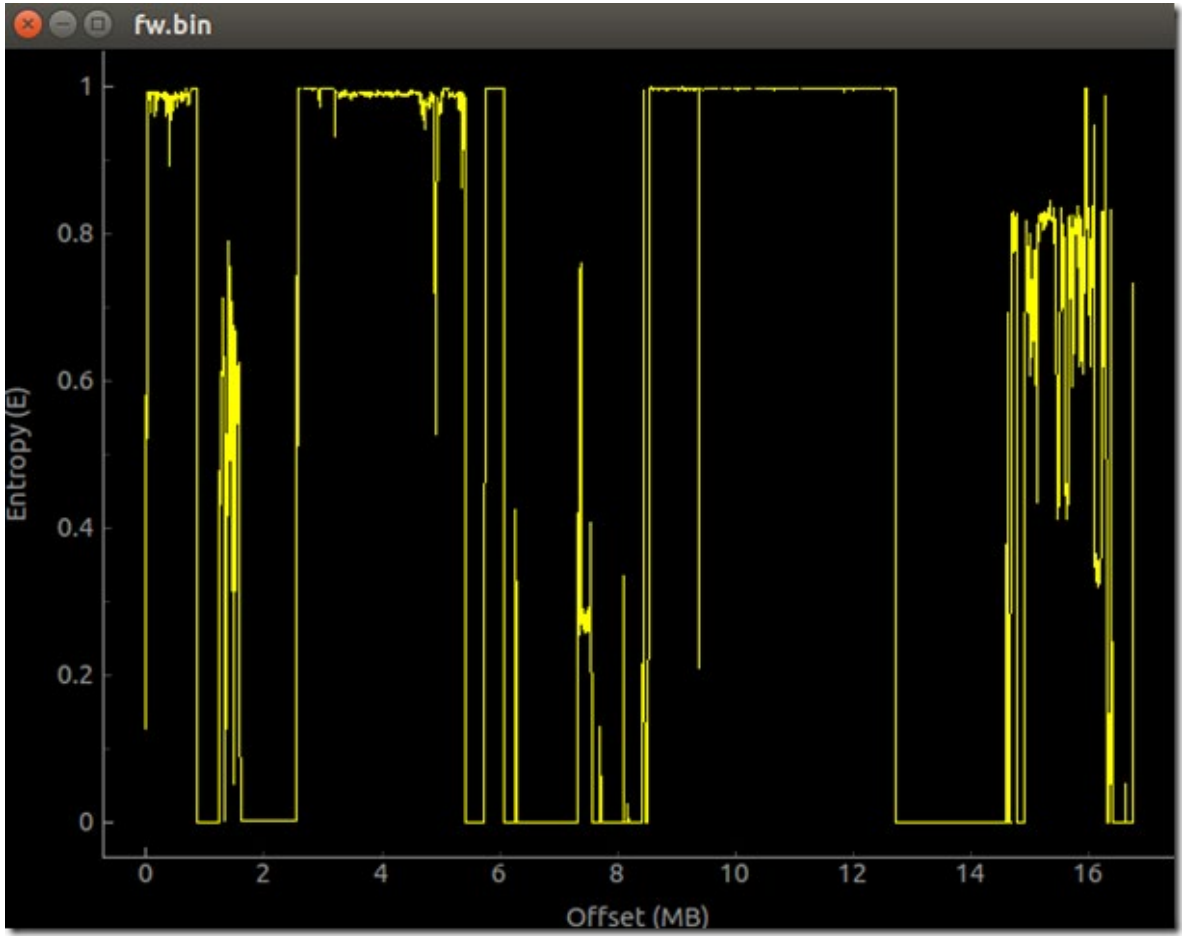
(example in next slide)

- If the firmware is encrypted, Firmadyne is not able to help you with.

For example :Binwalk –E ${Firmware_name}



The entropy of the encrypt or compressed firmware



The entropy of the raw data form firmware
(Maybe partial encrypted or compressed)

Some advices for environment setup:

Install the binwalk with the binaries compiled from the source code instead of using pip install binwalk.

If your Linux distribution is Ubuntu(>20.02), install python-is-python3 to let the module, Pexpect happy.

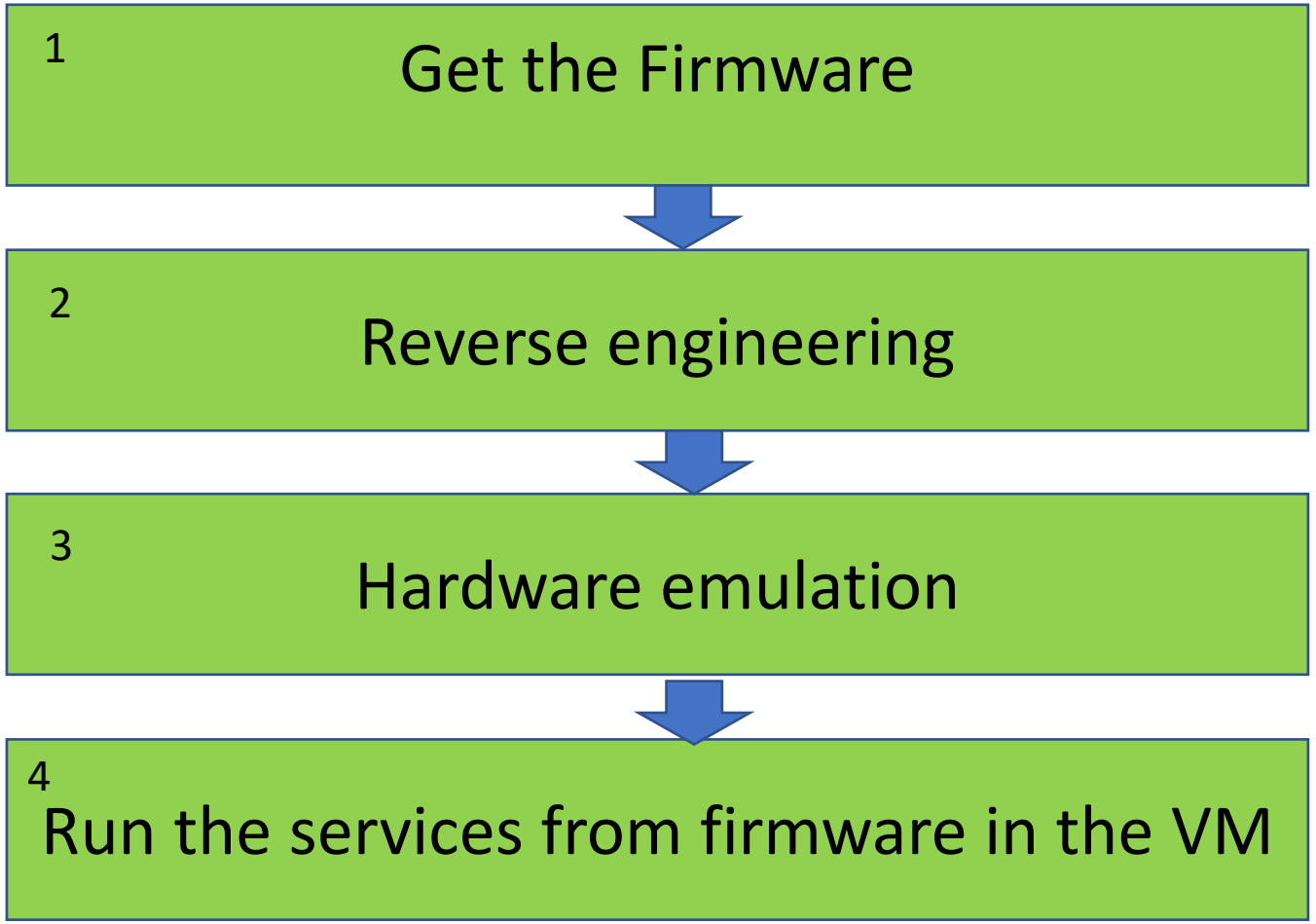
Remember to give the permission to user “Firmadyne”, let it able to write to database.

Some dependencies for python2 is not supported anymore, you can just remove such package inside setup.sh.

Remember to edit the config file for both Firmadyne and Firmware-analysis-toolkit.

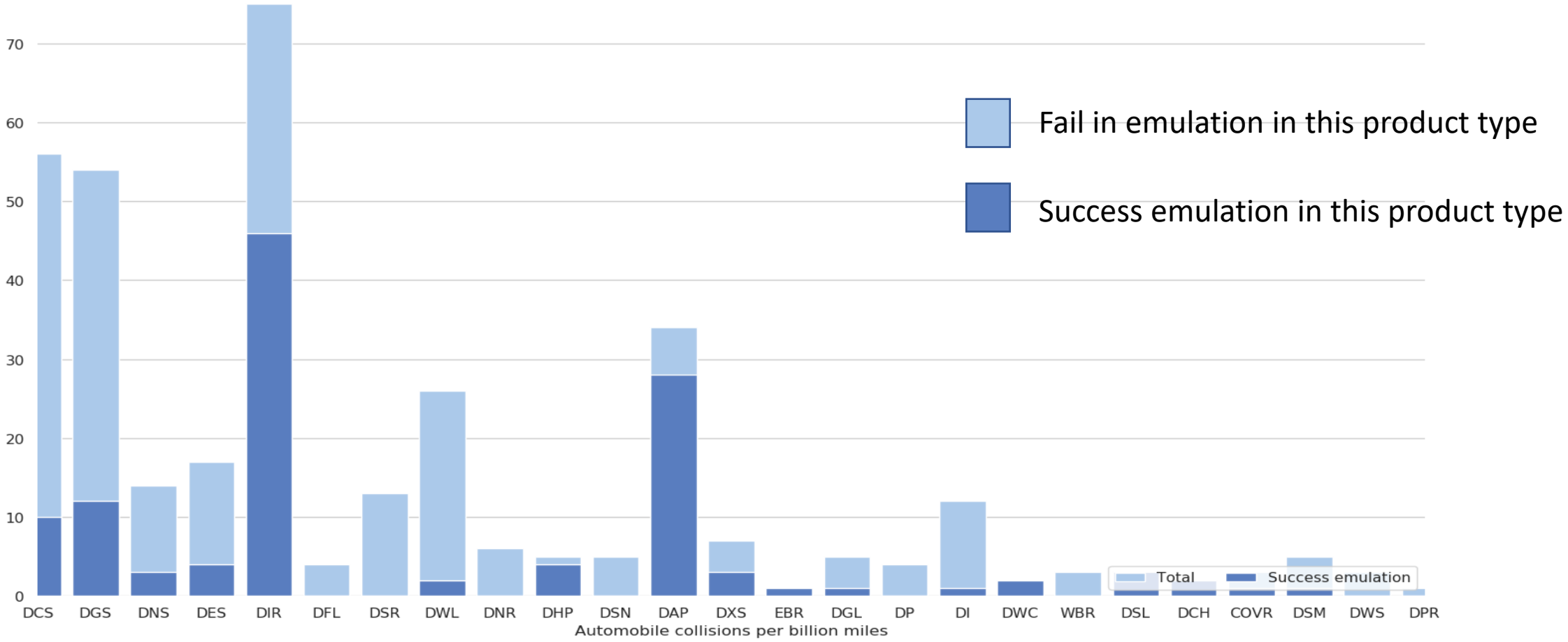
Maybe someone will struggle in environment setting, these will give a little help.

- So we have covered everything, we can automatically emulate the firmware, is this talk end here?
- It is worth to mention not every firmware can be emulated with the above process.



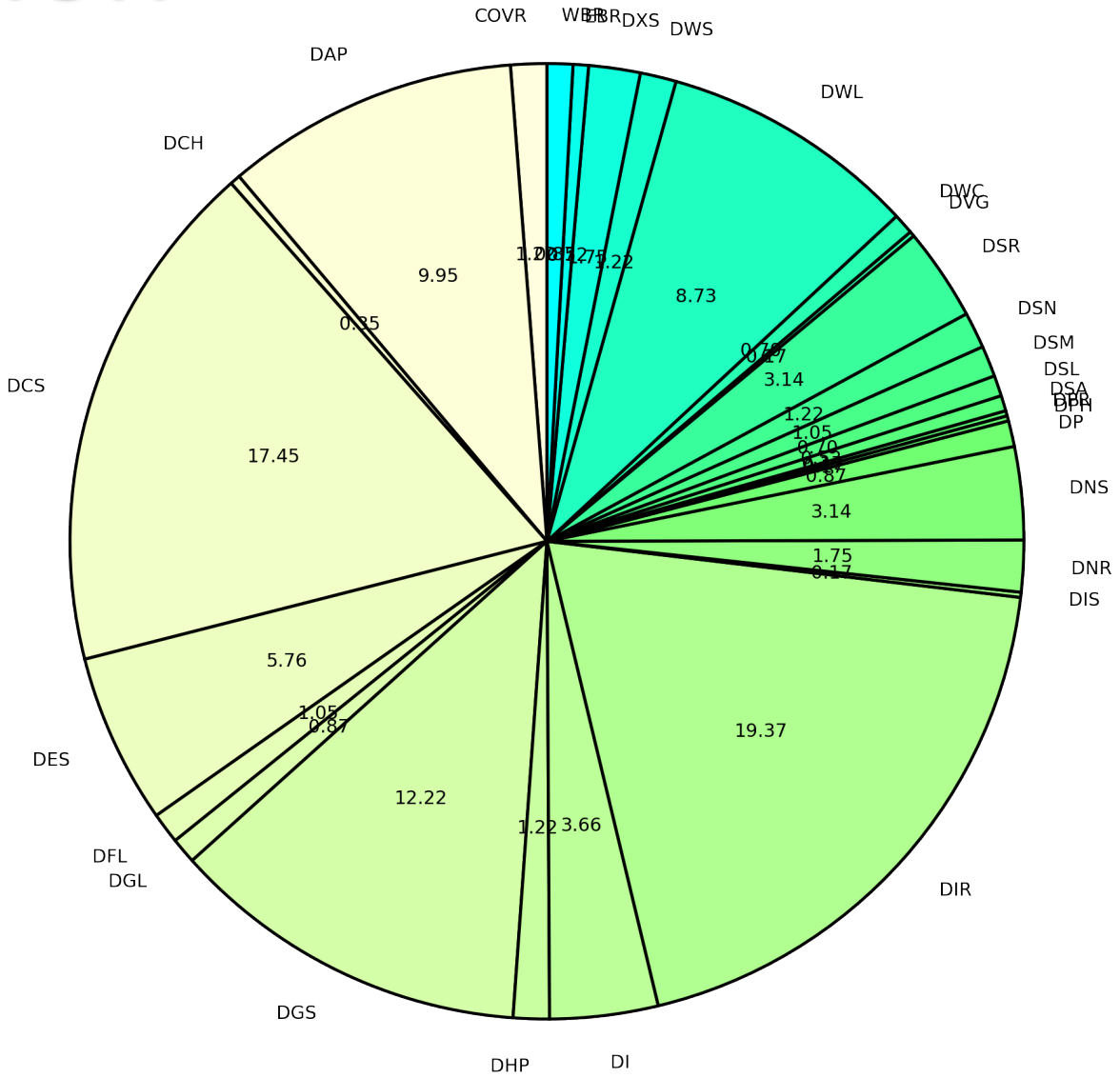
The Limitation and Challenge we discovered

Here is success emulation rate in whole firm wares from D-Link



More information

D-Link product distribution.
(dataset collect from the ftp server setup in different countries)



Firmadyne have some limitation like

- Only support the specific Linux kernel version (MIPS: v2.6.32, ARM: v4.1, v3.10), incase the firmware use the kernel with different version, the emulation will likely fail.

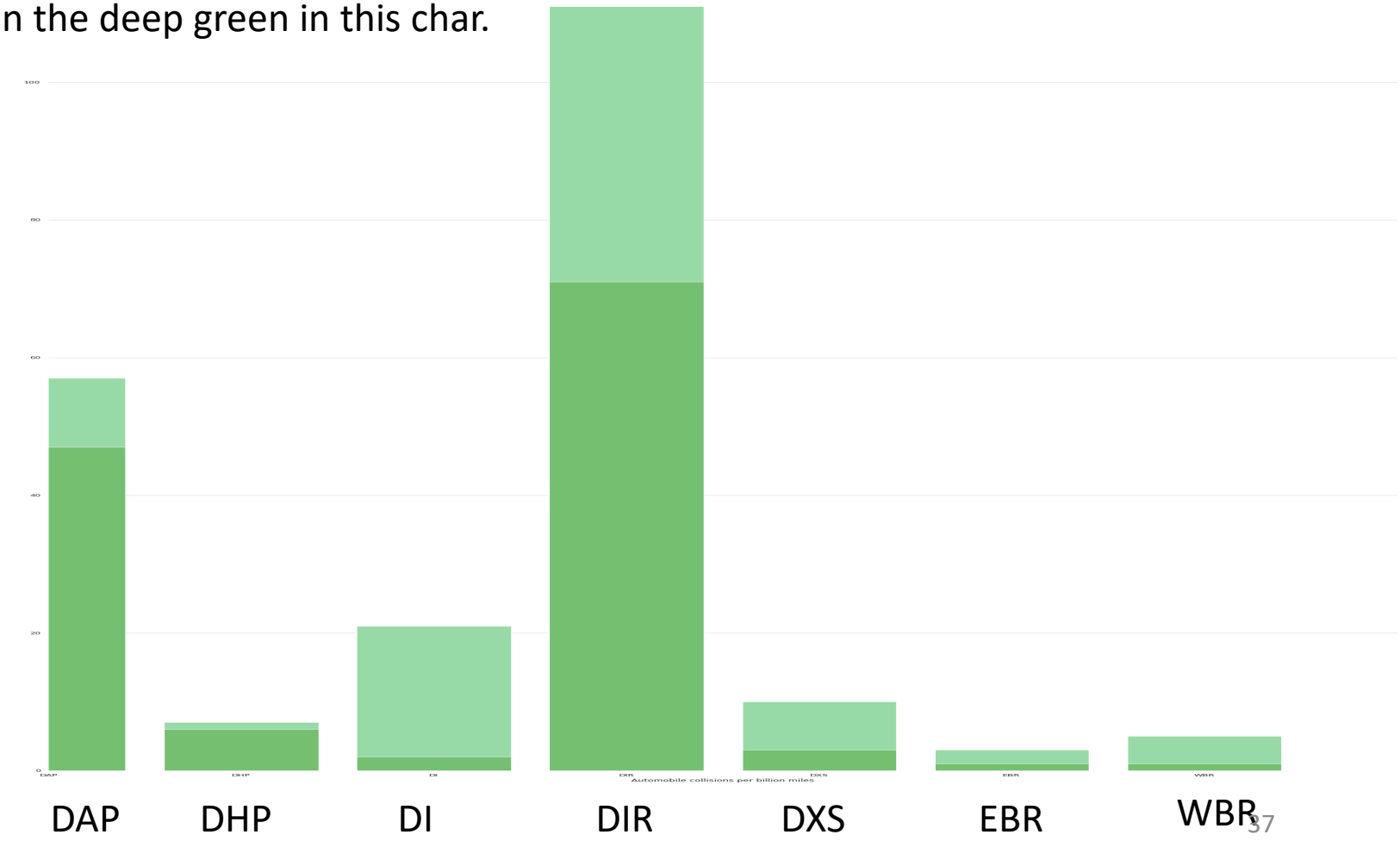
- Occasionally, driver will be not load successfully, due to the module is missing or dependency problem, it will need to be fixed manually.

- The inference of network information is not always generated correctly, fail in this step will lead to hard in conduct fuzzing, I will illustrate this problem with the figure in next slide.

The following chart shows firmware can be emulated correctly, and network interface also must be accessed unmistakably.

- Success emulation(boot up) ≠ inference of network setting is correct ≠ driver was loaded correctly.
- Everything setup correctly represent in the deep green in this char.

	Type	Total	Success
1	DAP	57	47.0
8	DHP	7	6.0
9	DI	21	2.0
10	DIR	111	71.0
26	DXS	10	3.0
27	EBR	3	1.0
28	WBR	5	1.0



How to eliminate such Limitation.

Problem analysis

My custom Autocheck program

Try to identify which key factors that affect the results significantly by collect the different figure in firmware

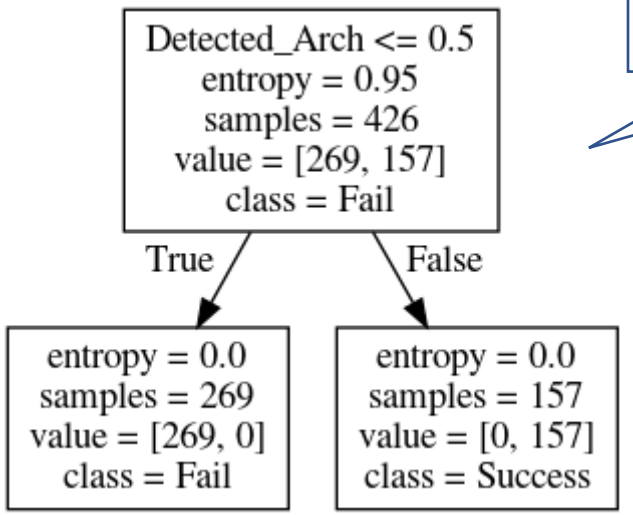
		Model	Revname	Firmware	Url	Date	Type	Filesystem	Success_e	Arch	Firmware	Network_	Services_	ip	Scan_info	have_beer	remark	kernel_version
2	0	COVR-22	A	1.04	ftp://ftp.dli	2019/9/17 00:00	COVR	None	0	None	COVR-22	0	0		None			
3	1	COVR-39	A	1.01b05	ftp://ftp.dli	2019/6/11 00:00	COVR	Squashfs f	0	None	COVR-39	0	0		None			
4	2	COVR-39	A	1.01	ftp://ftp.dli	2017/9/15 00:00	COVR	Squashfs f	1	armel	COVR-13	0	0		None		able to emulte with my custom arm kernel and	
5	3	COVR-39	A	1.01	ftp://ftp.dli	2017/9/15 00:00	COVR	Squashfs f	1	armel	COVR-26	0	0		None			
6	4	COVR-C	A	1.08	ftp://ftp.dli	2020/2/24 00:00	COVR	None	0	None	COVR-C	0	0		None			
7	5	COVR-C	A	1.08	ftp://ftp.dli	2020/2/24 00:00	COVR	None	0	None	COVR-C	0	0		None			

Problem analysis

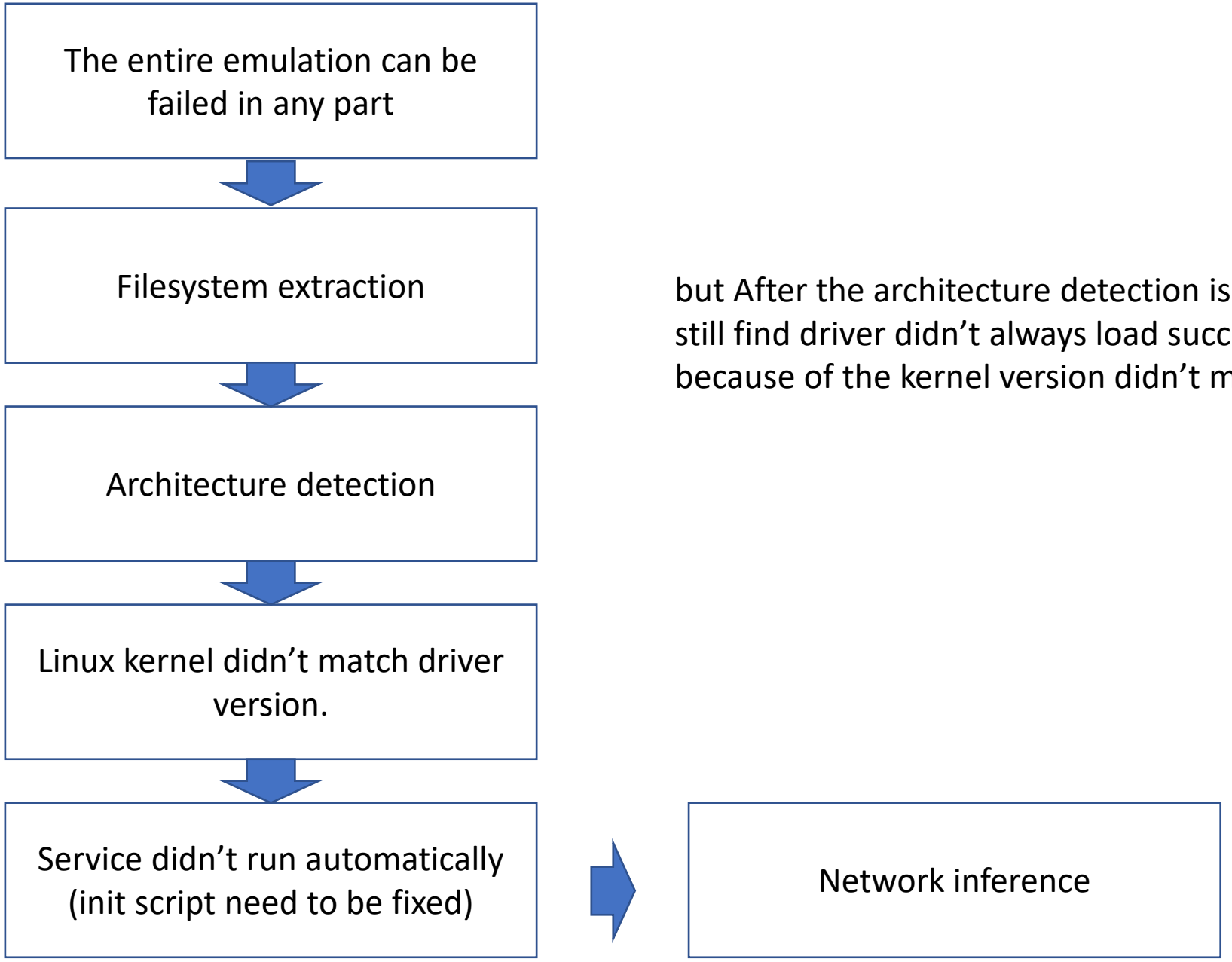
Using Random forest tree to analysis result

Decision tree can be visualized and tell us what the model have learn.

The result show us, once the architecture have been identified, the success rate will be 95%.



What is the key factor to achieve a success emulation ? Only one?



but After the architecture detection is solved manually, we still find driver didn’t always load successfully typically because of the kernel version didn’t match driver version.

This is how we check the linux version from firmware issues.

From our customer

Our custom environment that Try our best to fit the firmware originally should be.

Are we able to check the original kernel version from firmware?

```
$ strings ./img |grep “Linux version
```

DAP-3662_REVA_FIRMWARE_1.05beta6-rc082	b'Linux version 2.6.31--LSDK-10.2.85 (release@cdWSCMPL06) (gcc version 4.3.3 (GCC)) #1 Fri Nov 9 11:31:21 CST 2018\n'
DI-524_REVC_FIRMWARE_3.[illegible].BIN	None
DCS-1000W_REVA_FIRMWARE_1.30.BIN	None
DIR-636L_REVA_FIRMWARE_1.0[illegible].BIN	b'Linux version 2.6.36-svn5117 (root@localhost.localdomain) (gcc version 3.4.2) #350 Thu Apr 11 19:00:47 CST 2013\n'
DAP-1520_REVA_FIRMWARE_1.08.BIN	b'Linux version 2.6.36.x (root@IVAN-800G-laptop) (gcc version 3.4.2) #3 Thu Apr 21 15:31:25 CST 2016\n'
DAP-2660_REVA_FIRMWARE_1.15RC093.bin	b'Linux version 2.6.31--LSDK-10.2.85 (release@cdWSCMPL06) (gcc version 4.3.3 (GCC)) #1 Fri Jul 29 13:44:49 CST 2016\n'
DCS-5010L_REVA_FIRMWARE_1.16.01.BIN	b'Linux version 2.6.21 (andy@ipcam-linux.alphanetworks.com) (gcc version 3.4.2) #2494 Fri Jan 25 17:20:38 CST 2019\nLinux version \n'
DAP-3662_firmware_v2.01r098.bin	b'Linux version 2.6.31--LSDK-10.2.85 (release@cdWSCMPL06) (gcc version 4.3.3 (GCC)) #1 Thu Jan 2 11:26:16 CST 2020\n'
DAP-1320_REVA_FIRMWARE_1.20B07BETA03	b'Linux version 2.6.31 (root@CK-ubuntu) (gcc version 4.3.3 (GCC)) #1 Wed Jun 18 11:07:30 CST 2014\n'
DIR-826L_REVA_FIRMWARE_1.04.BIN	b'Linux version 2.6.36+ (root@localhost.localdomain) (gcc version 3.4.2) #351 Thu Apr 11 19:13:22 CST 2013\n'

For those which can not be identify, the rootfs were not create in Linux environment, figure out by look at the file's information.

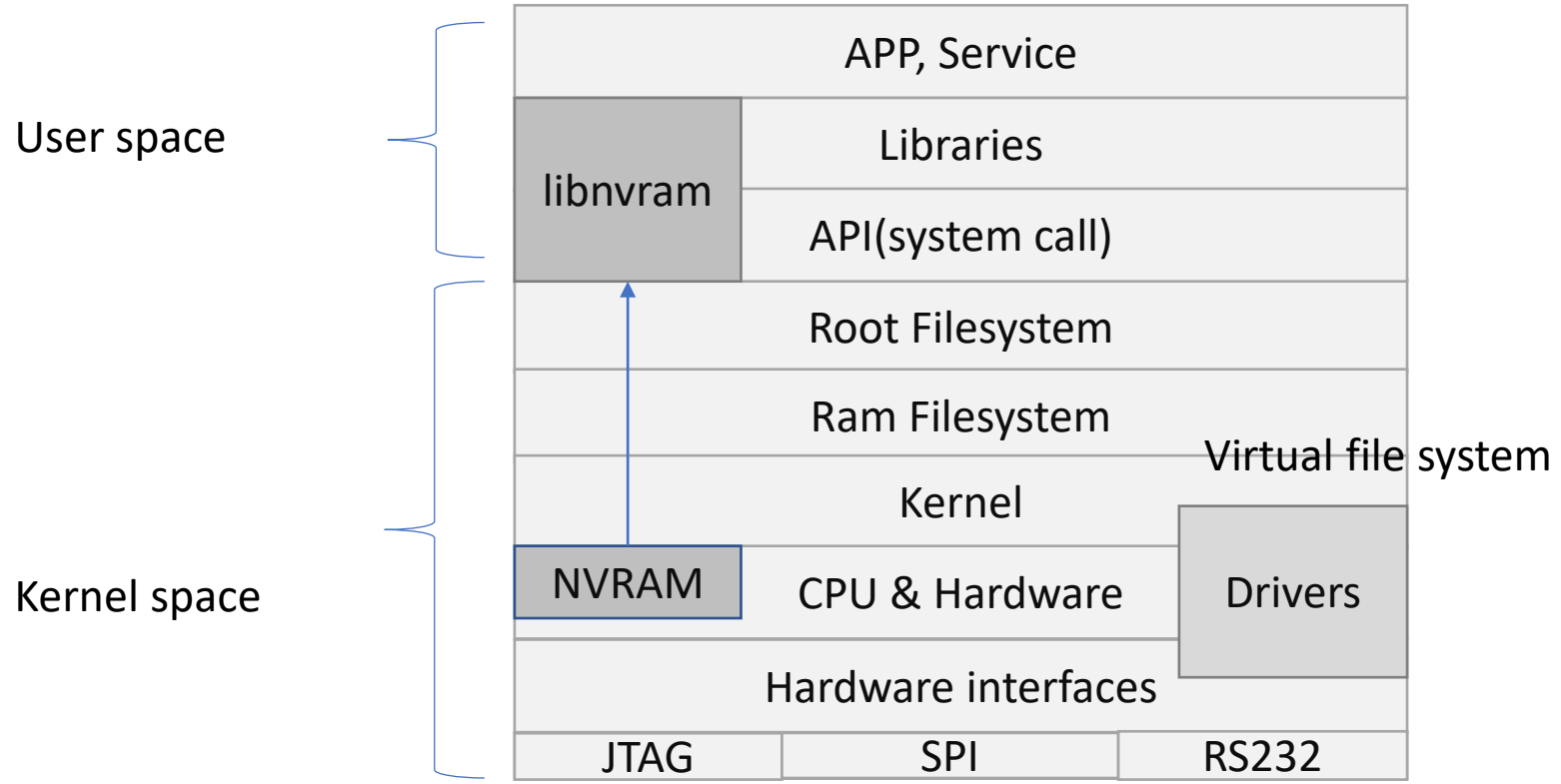
What we do to fix the firmware ?

Architecture of firmware

Boot process of firmware

① UEFI
② Boot loader
③ Linux kernel
④ RamFS
⑤ Init (PID 1)
⑥ RootFS
⑦ Init (PID 1)
⑧ /etc/rcS (inittab)
⑨ Services

Rebuild the entire step with the information infer from original framework.



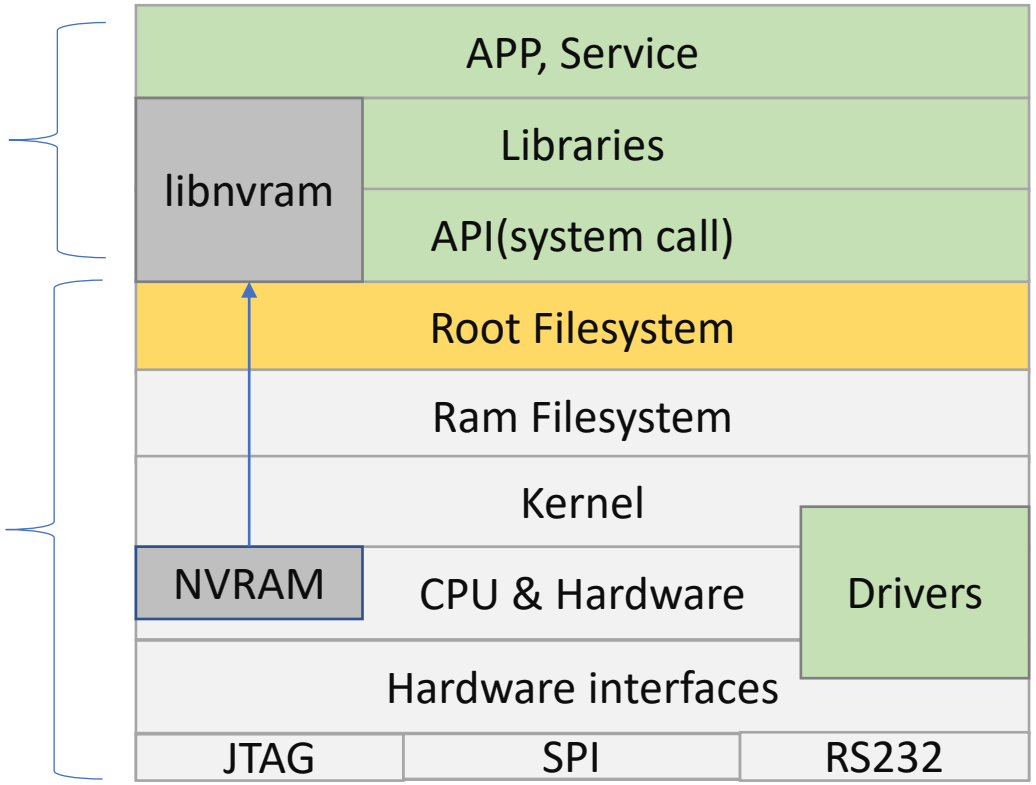
How does the approach O work?

Components extracted from firmware

We discard the kernel and Ram filesystem from original firmware.

The part we take from original firmware

Something must be patched or to ensure emulation works fine.

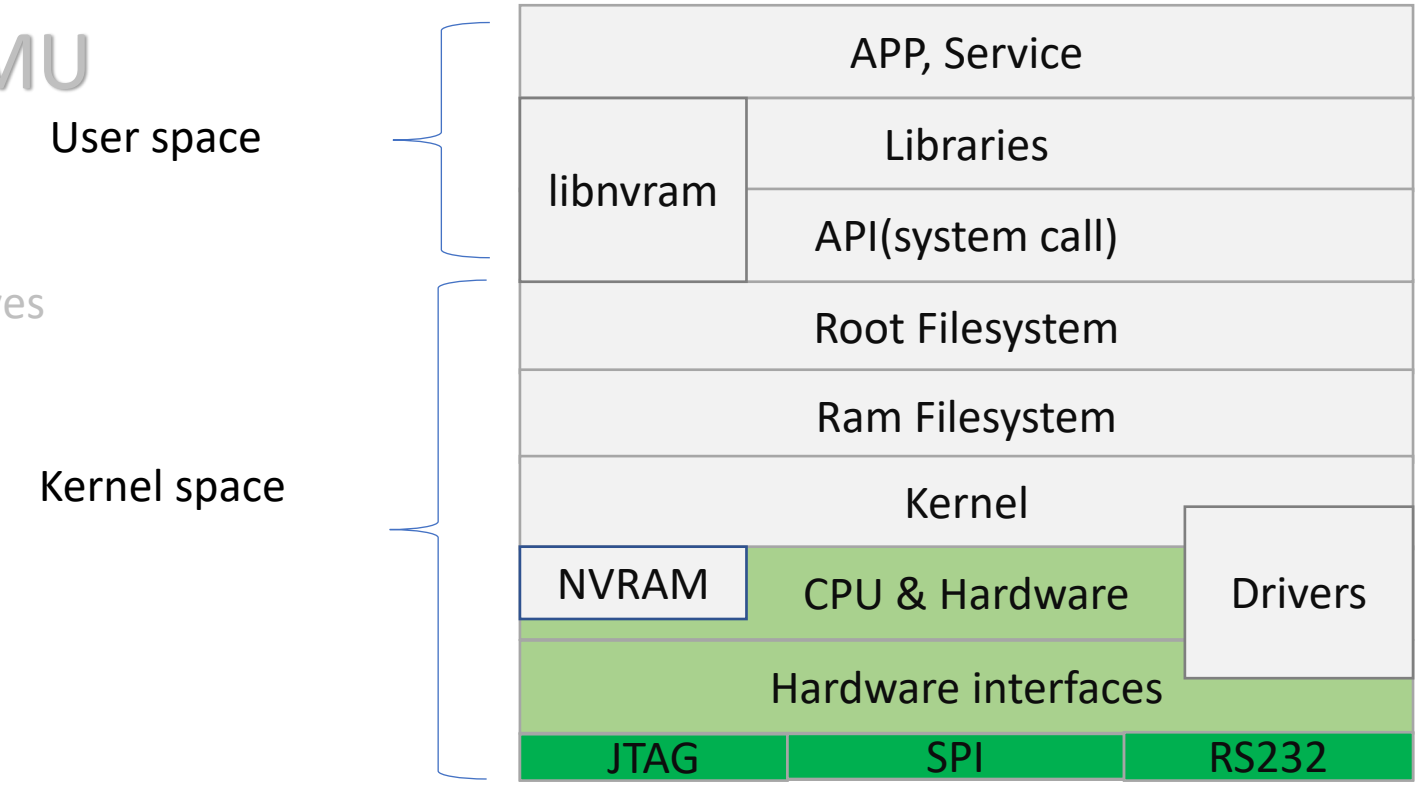


How does the approach O work?

Components That emulate by QEMU

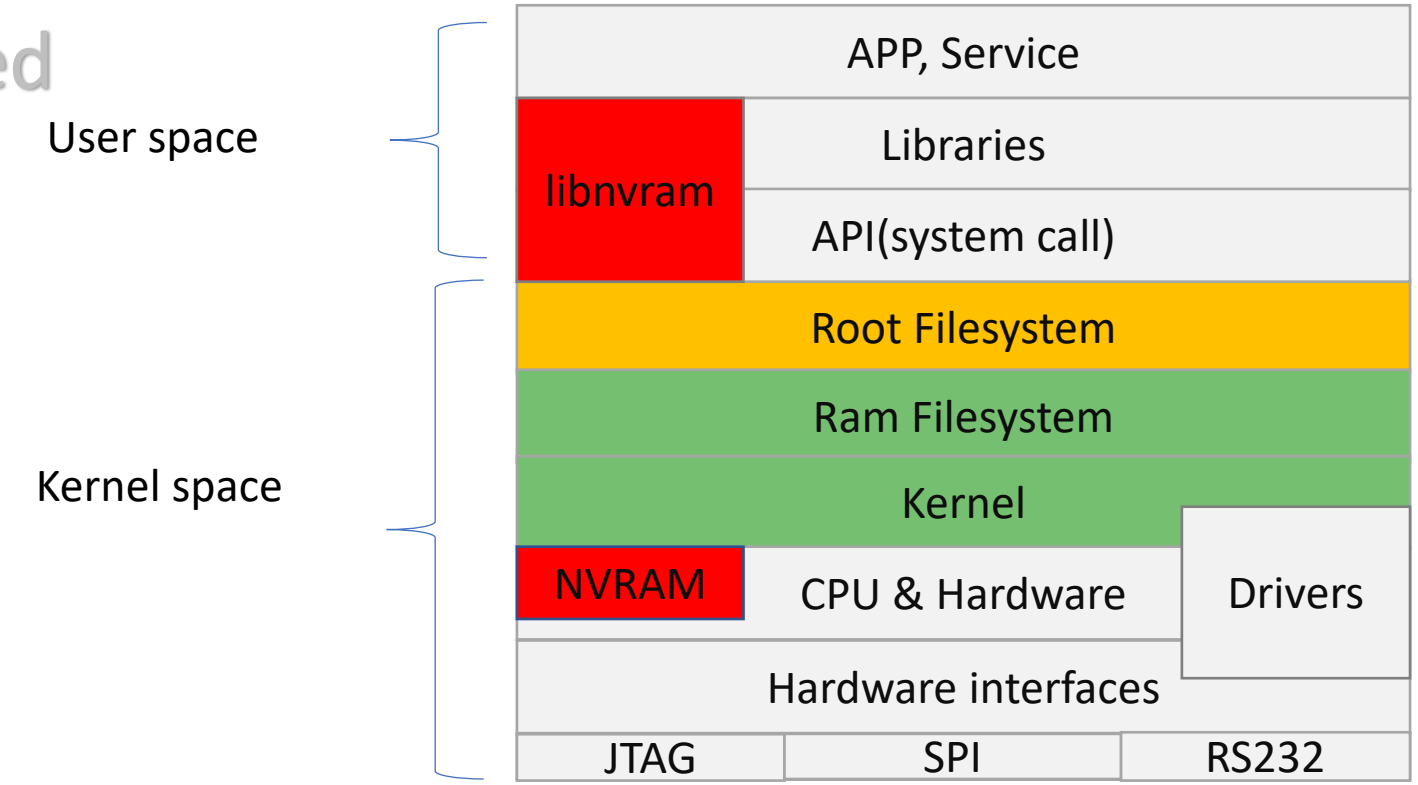
We will custom those component by ourselves

- QEMU work completely fine with this component, but still need to be configured manually. (choose the cpu architecture and machine type)
- QEMU work completely fine with this component.



How does the approach O work?

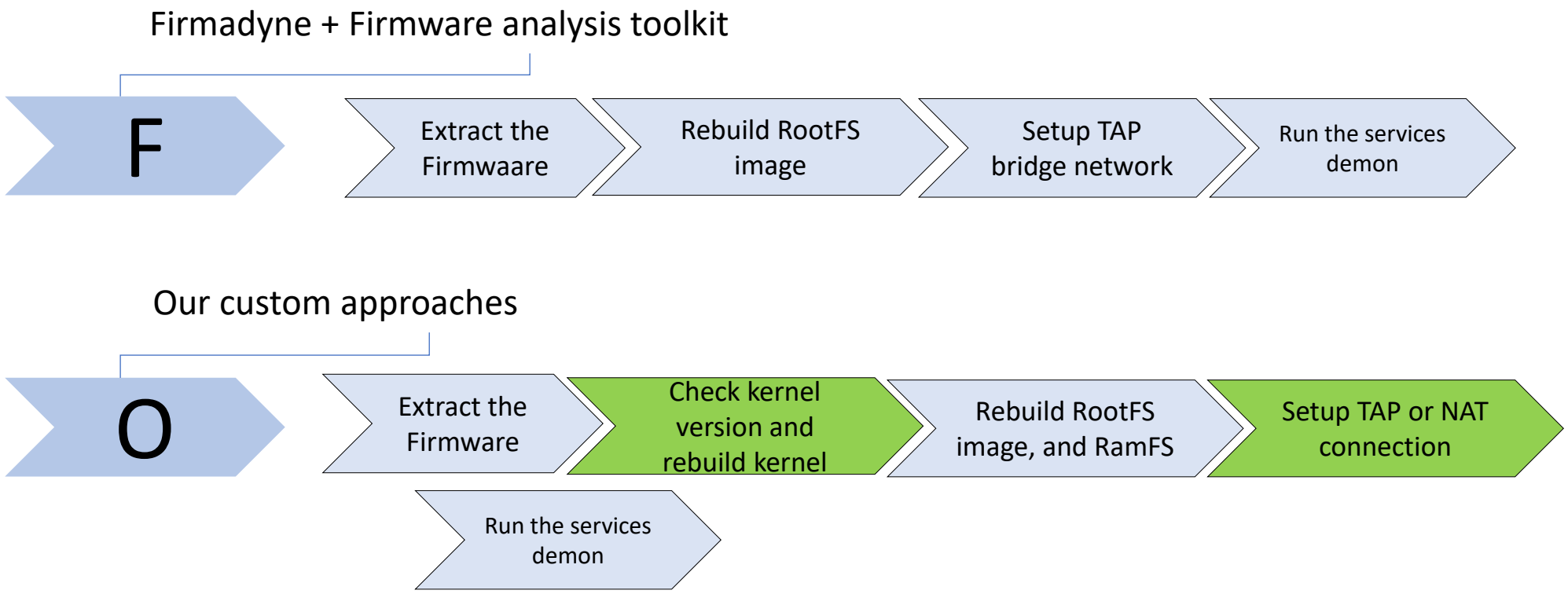
Components that can be emulated



- The part we recompile with the information infer from DUT
- We will patch something for dynamic analysis
- This part is not in firmware, we must use kernel hook or user mode hook to deal with such problem

Conclusion: The difference

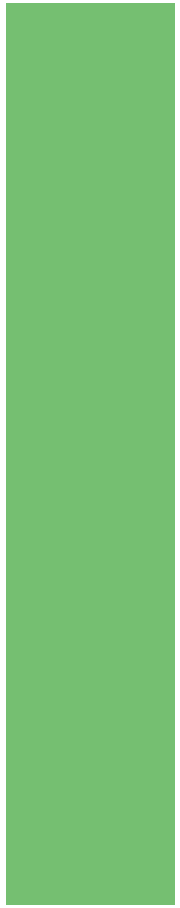
Two approaches:
F for Firmadyne & O for our approach



When we encounter this situation (low emulation rate), We consider how to improve the success rate in these product type

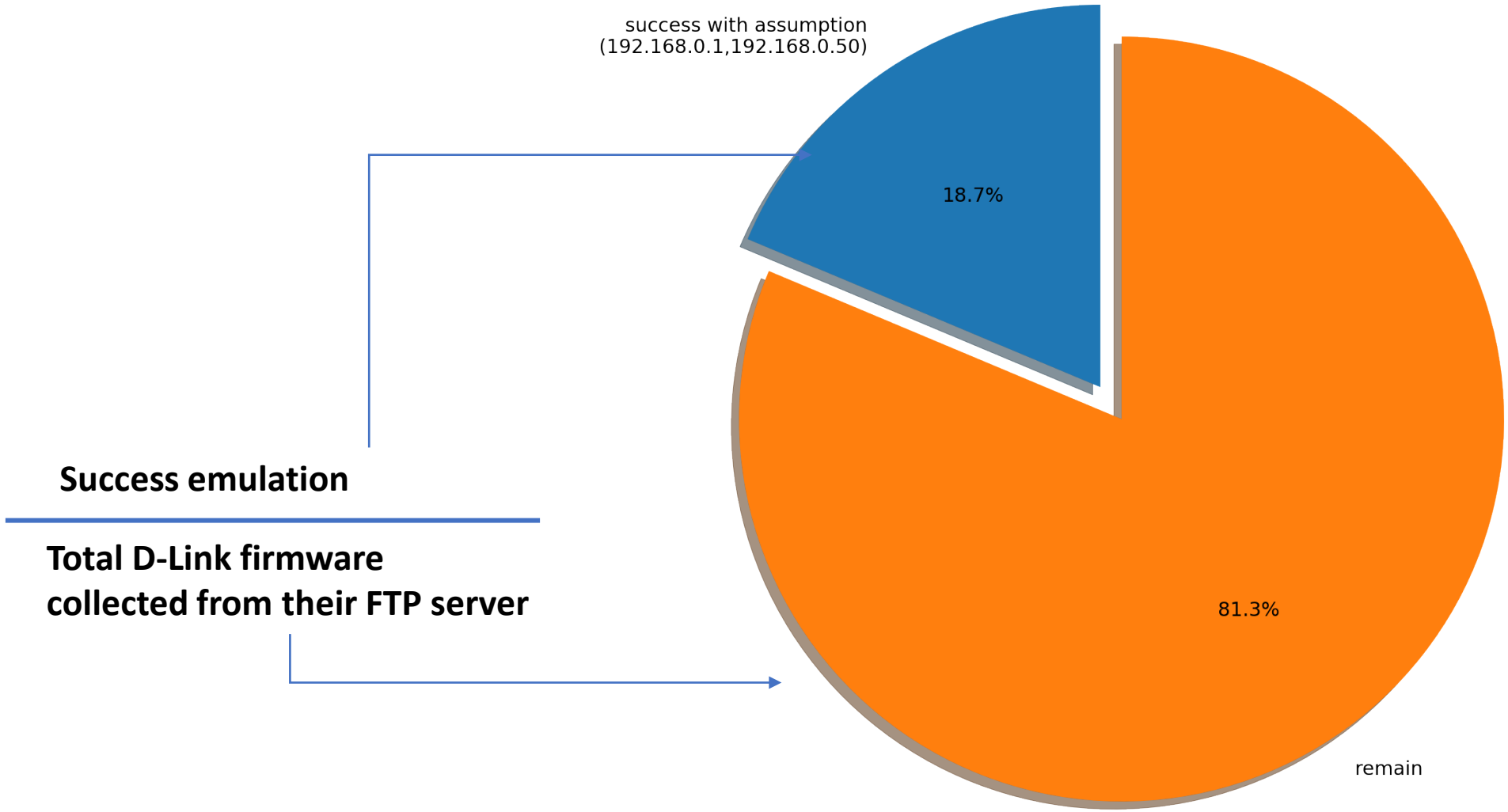
For example:
We want the bar like this.

After our improvement



Become this

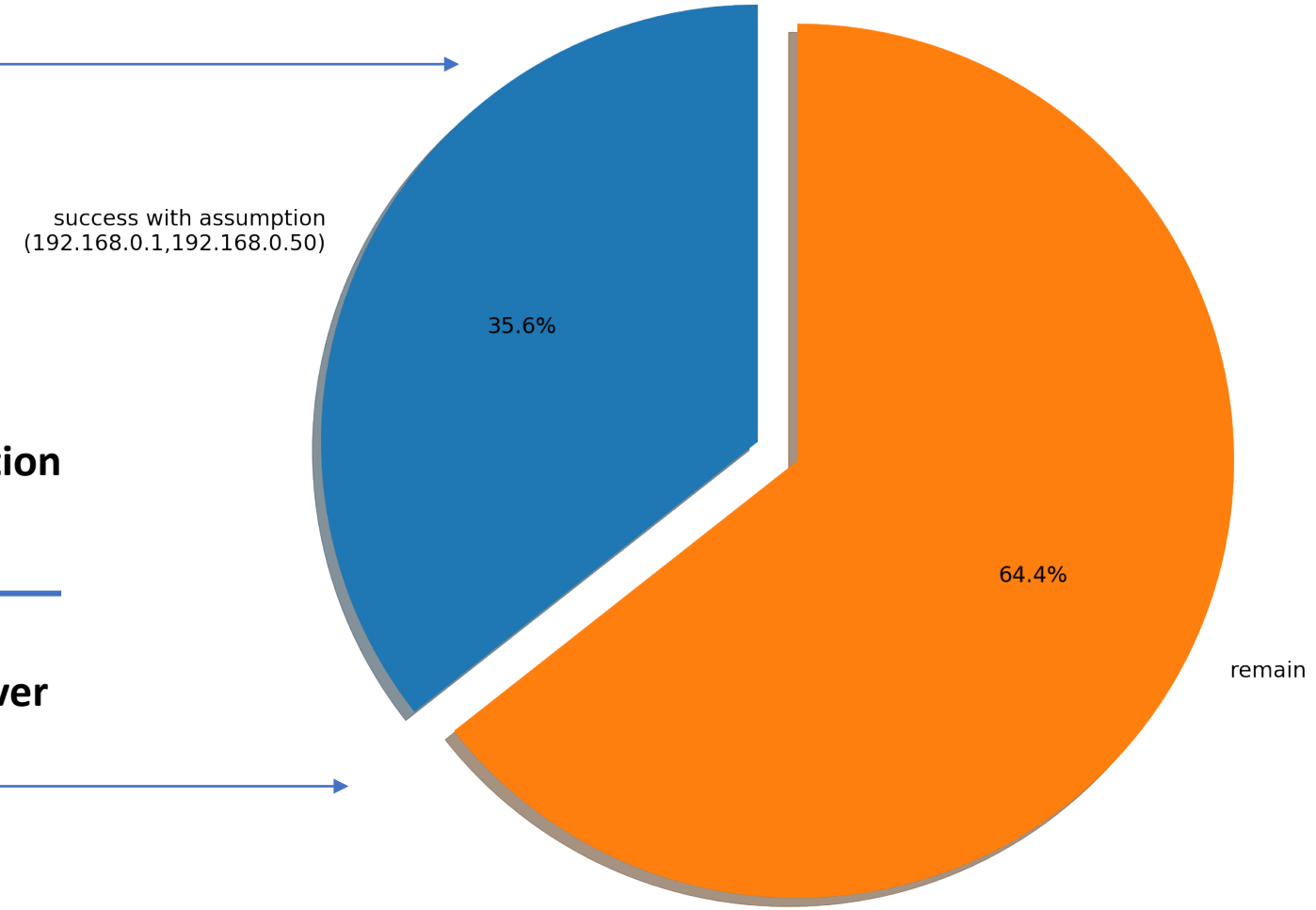
Original


success with assumption
(192.168.0.1,192.168.0.50)
18.7%
81.3%
remain
Success emulation
Total D-Link firmware
collected from their FTP server

After

Success emulation in assumption
In our approaches.

Total D-Link firmware
collected from their FTP server



Future works

The problems still need to be solved.

F	O
Only Linux 2.6 kernel	Building environment for old kernel hard to be represent (cross compile)
Limit architecture and environment	NVRAM emulation
License problem	RamFS initialize problem
	Module compatibility
	Figure out why the kernel that extracted from firmware can not be loaded by QEMU

Method comparison

F= Firmadyne + Firmware analysis tool kit
O= Our approaches.

	Coverage	NVRam emulation	Linux kernel version	Networking	Custom RootFS. (busybox, analysis tool)	Architecture & SoC type	Degree of automation
F	As benchmark	Yes	2.6	TAP (Cannot be simulated at once)	Can be edit menually.	mipeb, mipsel, ~~armel~~, specific SoC type	Better for now
O	Much much better	No for now.	Version can be dynamic, but the version number is greater the better for now.	TAP or NAT	RootFS image is created during the process.	There is no limit theoretically, up to QEMU emulator	Even the environment have been settled, the services in the firmware need to be start manually for now.

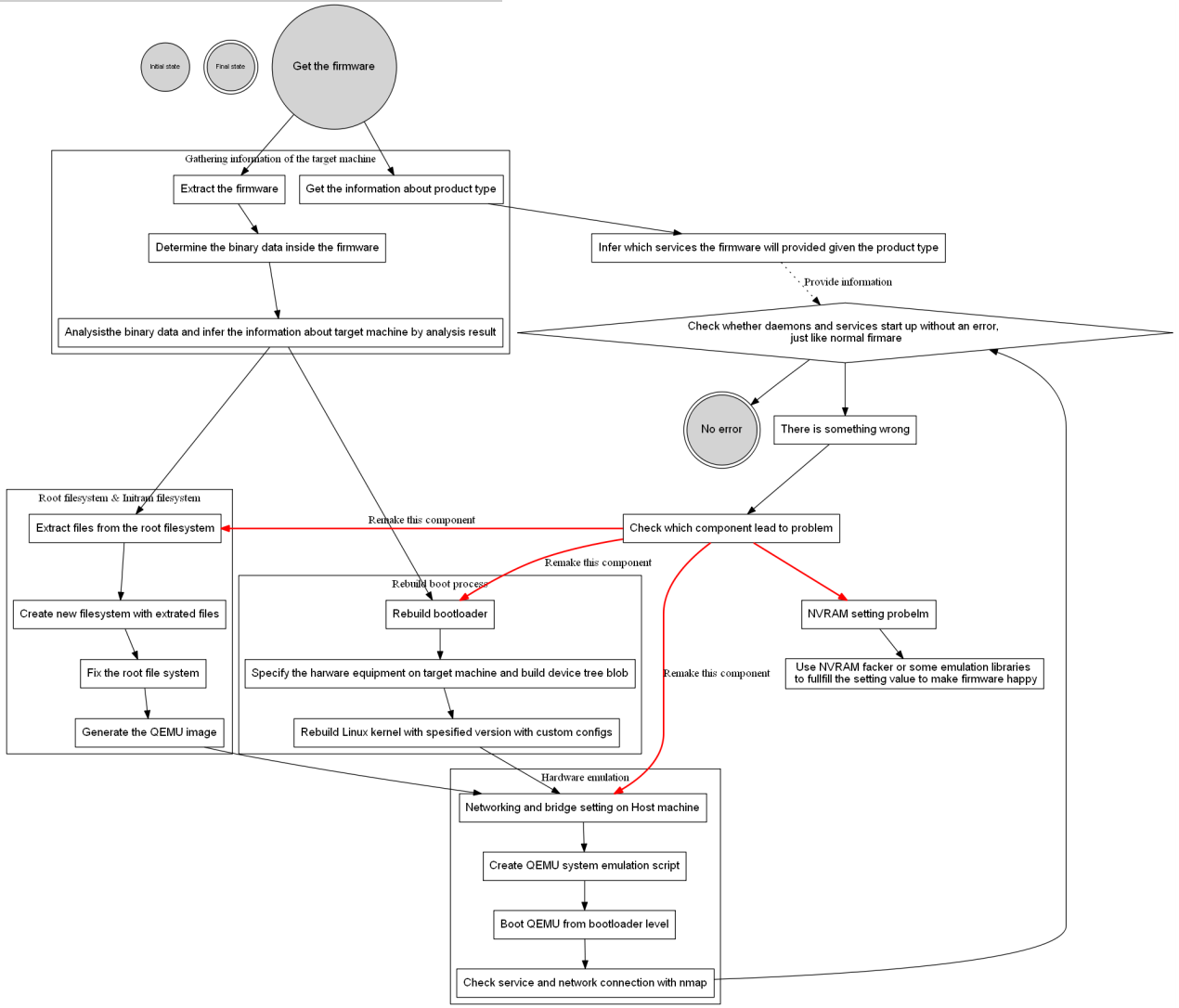
Method comparison

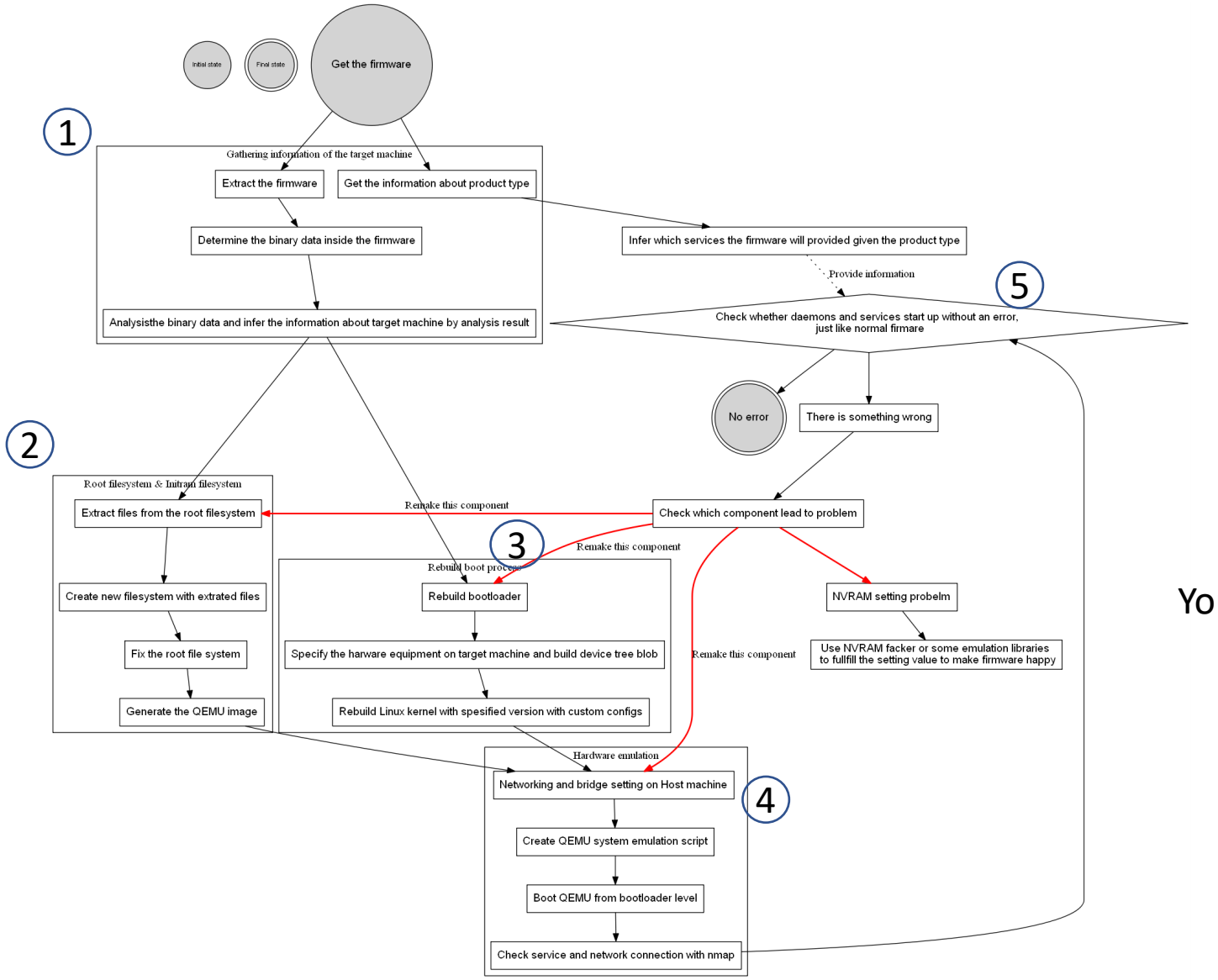
F= Firmadyne + Firmware analysis tool kit
O= Our approaches.

	Coverage	NVRam emulation	Linux kernel version	Networking	Custom RootFS. (busybox, analysis tool)	Architecture & SoC type	Degree of automation
F	As benchmark	Yes	2.6	TAP (Cannot be simulated at once)	Can be edit menually.	mipeb, mipsel, ~~armel~~, specific SoC type	Better for now
O	Much much better	No for now.	Version can be dynamic, but the version number is greater the better for nc	TAP or NAT	RootFS image is created during the s.	There is no limit theoretically, up to QEMU emulator	Even the environment have been settled, the services in the firmware need to be start manually for now.

We are able to use our approach to emulate the firmware from Adventech.

Flow chart of our method

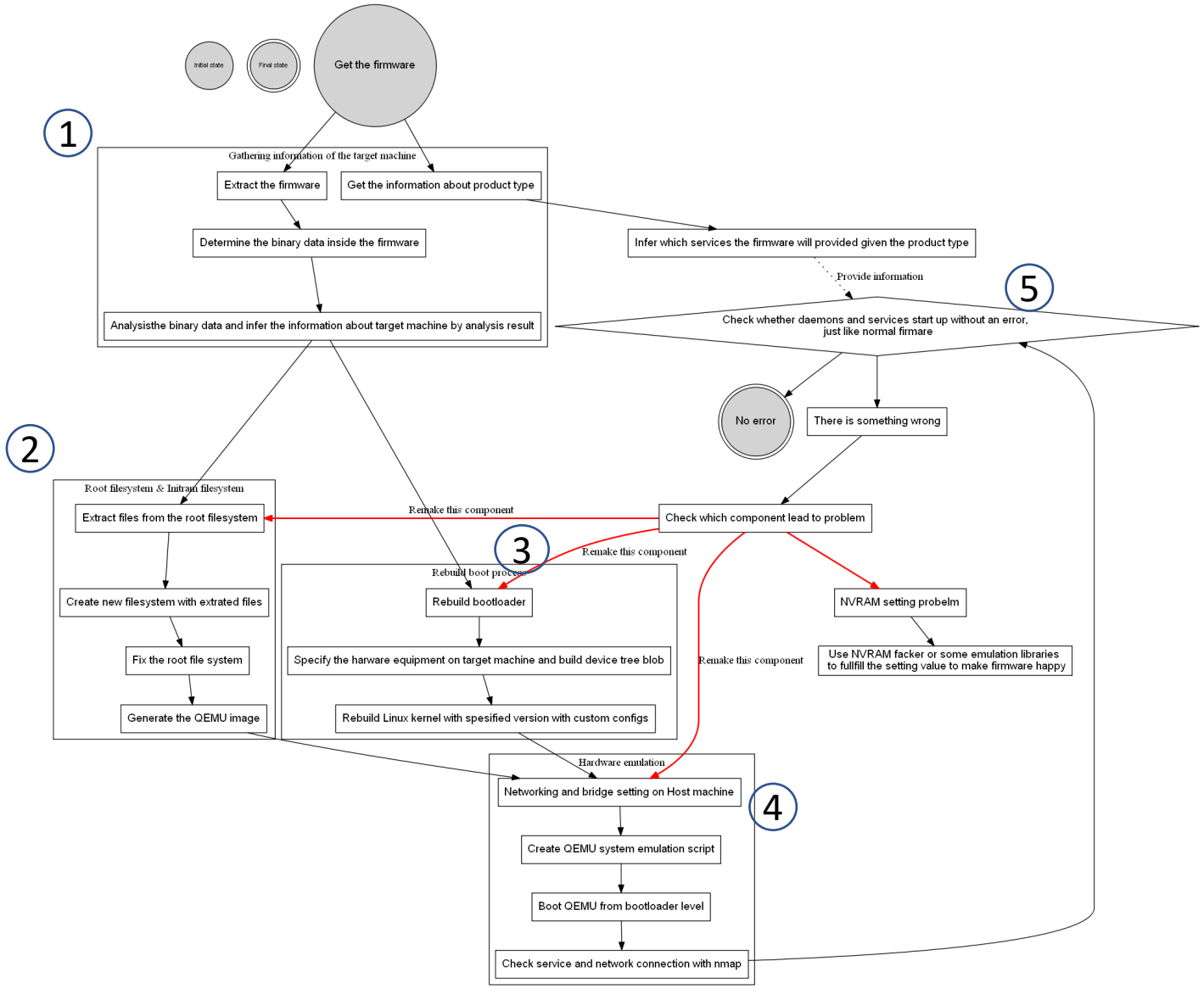




Flow graph

In detail version

You can check the detail step in appendix.



Flow graph

In detail version

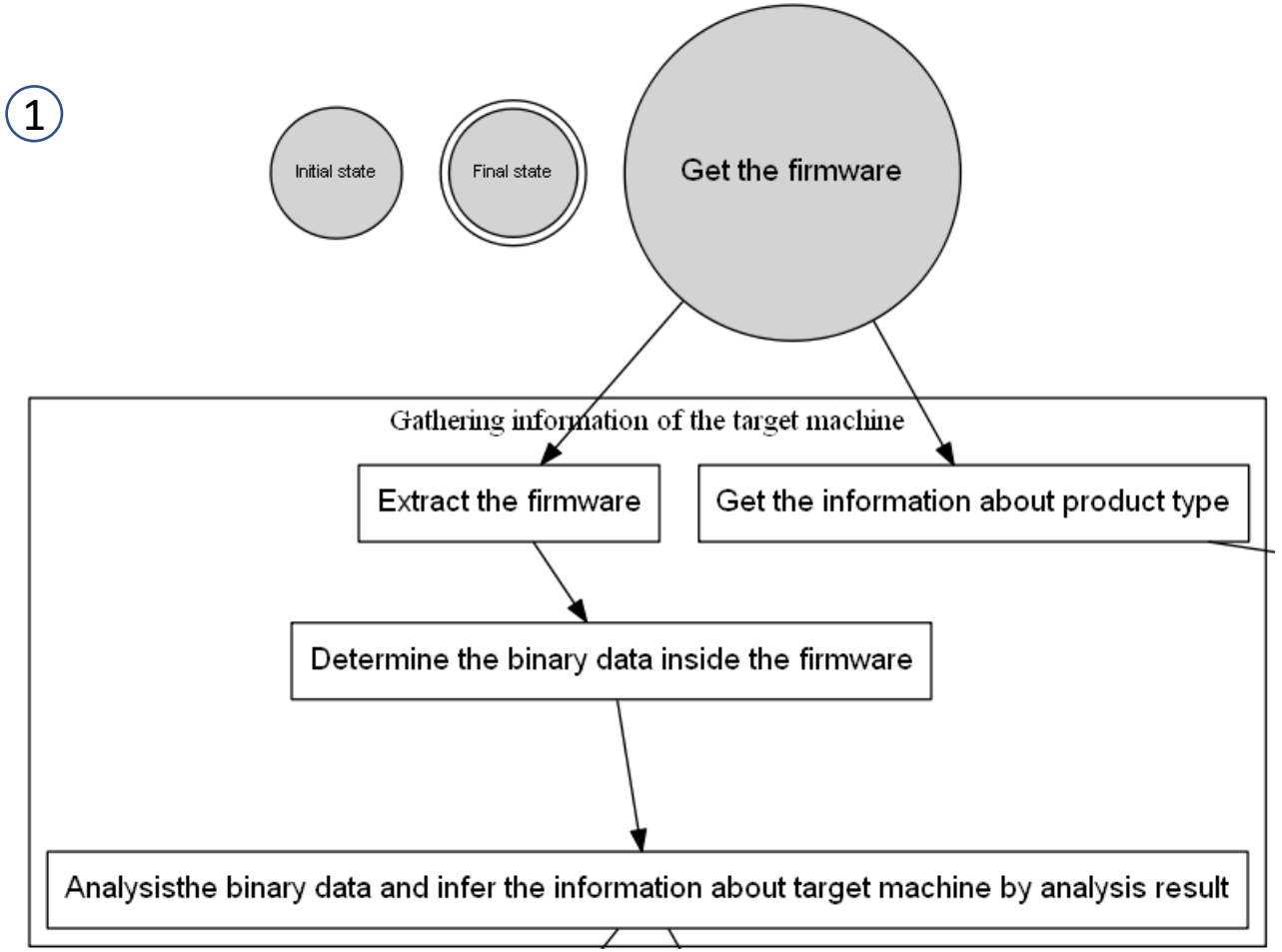
1. Gathering information from DUT.
2. Root Filesystem, Ram Filesystem rebuild.
3. Rebuild boot process.
4. Hardware emulation configure.
5. Check the network services.

- Thanks for your listening.

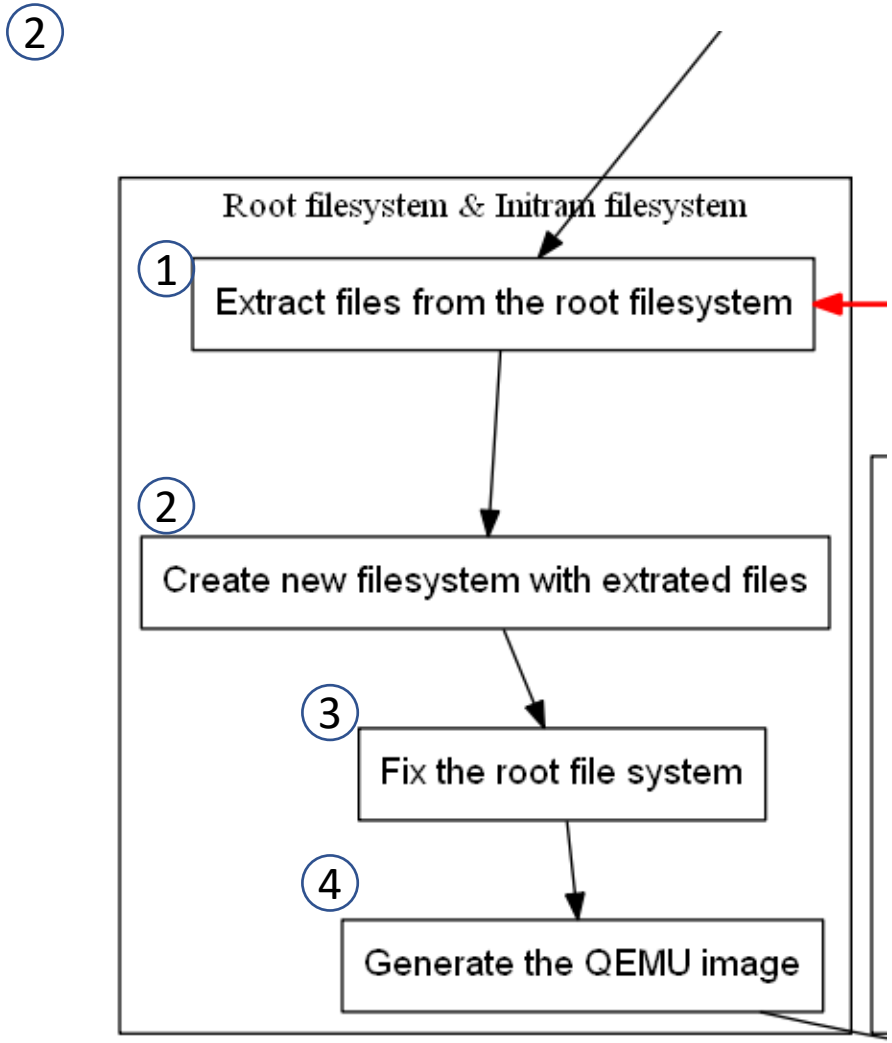
The following pages is the appendix of the flow chart.

Gathering information of the target machine

①



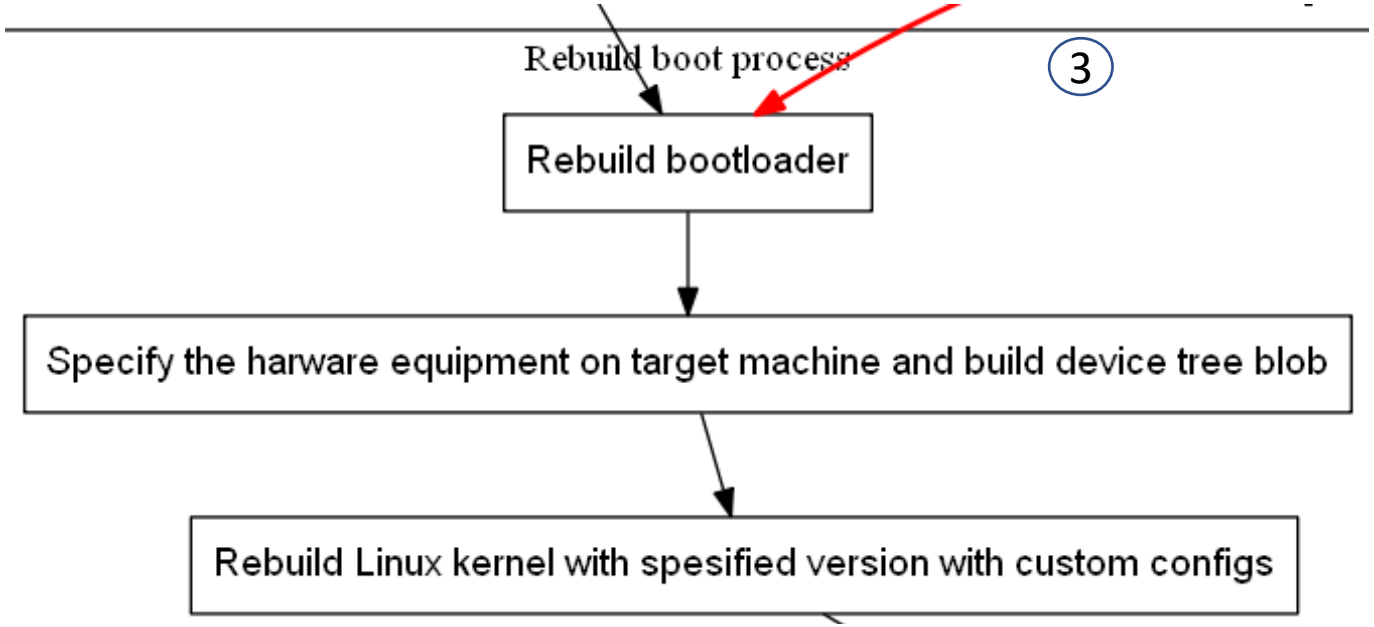
Rebuild filesystems

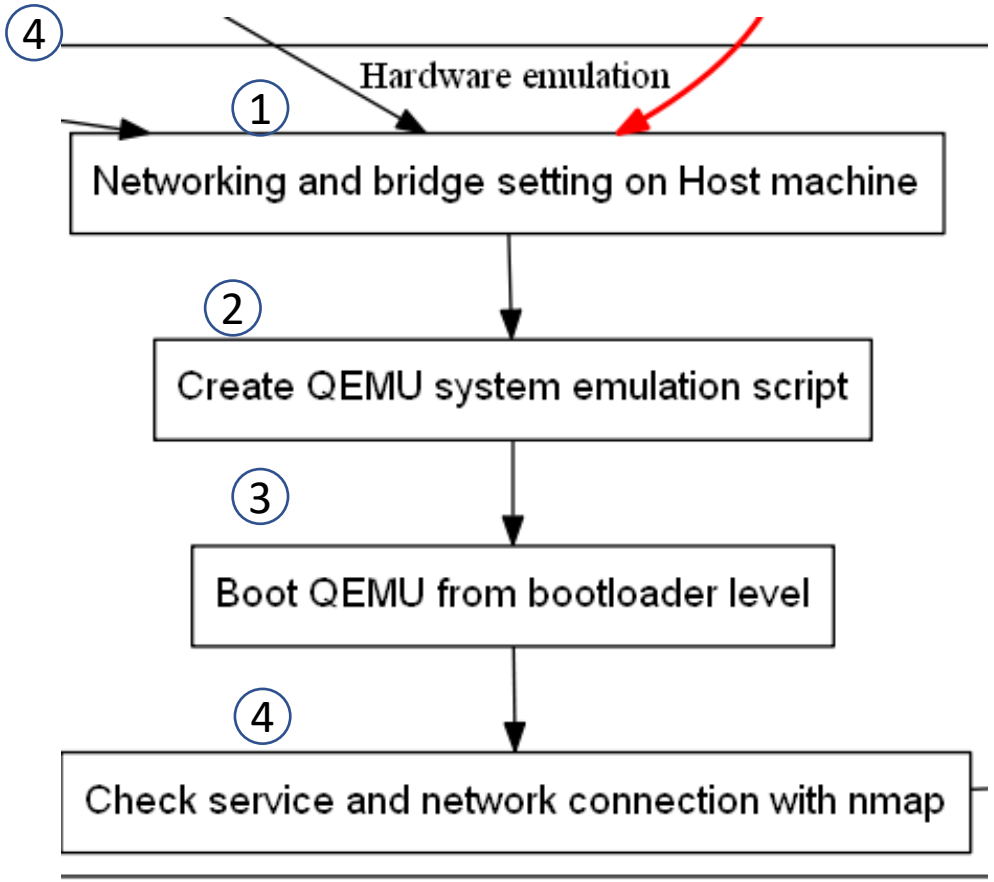


This stage consists of the following steps

2.1 Extract files from the root filesystem.
2.2 Create new filesystem with extracted files.
2.3 Fix the root file system
2.4 Generate the QEMU image.

Rebuild boot process





This stage consists of the following steps

4.1 Networking and bridge setting on Host machine
4.2 Create QEMU system emulation script
4.3 Boot QEMU from bootloader level
4.4 Check service and network connection with nmap

Check whether daemons and services start up without an error just like normal firmware