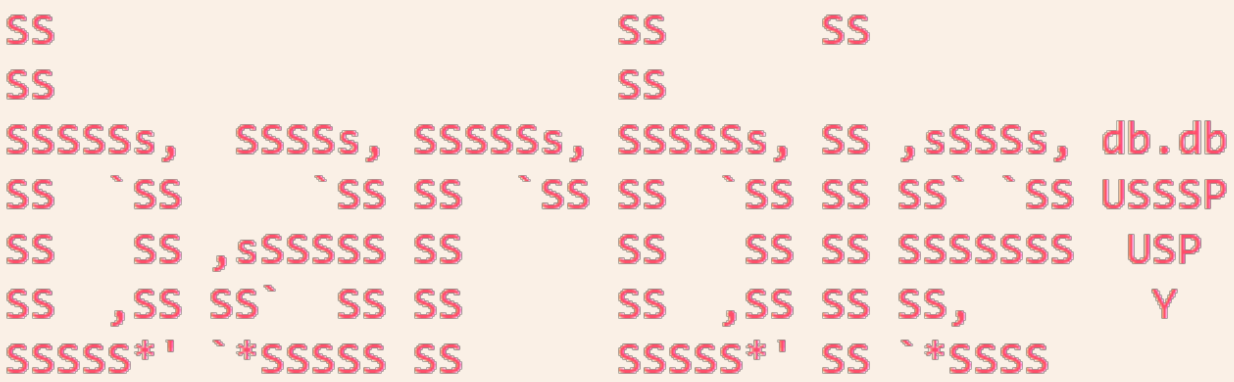

Firmware 101

2018-08-21 1.2k words 7 minute(s) firmware reversing

Getting the Code

Extracting the Firmware

In the last post, we discussed how to find important information about how to communicate with the device's. In this post, we are going to describe the standard approach of getting the code we want to reverse and use the information we collected before.

Goals

One of the most important things to think about when doing reverse engineering is to know what questions you want to answer. If you start looking for ANY kind of information, data or piece of code, you will end with lots of knowledge on stuff you have no idea what it is. Yes, you will drown in a lake of information! So, before starting trying to understand everything, ask the most important questions for you and focus on that. Try to keep your way, and check regularly that you are not deep in the mud.

I often use what I call “Jack the Ripper” approach: do it calmly, piece by piece and try to enjoy it :) Start looking for specifications online about the communication protocol, about the board itself, about the components, try to understand the undocumented part, double check the documented part. When you’re done, go for some critical system or code, like authentication algorithm or password generator / checker, anything that looks like it’s handling juicy data. Embedded systems security is not that advanced, but still all devices need some kind of identification mechanism. These IDs should be unique for each hardware, but at the same time generated in production. They should be able to authenticate somewhere and also be present in the firmware, which is supposed to be the same for all devices. Now, think about solving this problem for large scale production?

Talking with the system

The best part and the essential difference while reversing hardware is the bare metal in your hand. The electrical signals. It is the possibility of measuring everything and actually see how the data flows. Adding some LEDs in between never gets boring and it is beautiful every single time! If this fact alone make your heart beats a bit faster, same here! It is another kick, way stronger than a piece of code :) Without much information and with the aid of some cheap equipment, it is possible to get lots of information. We already know that with a simple logic analyzer, we can sniff the data traffic. With a more expensive lab however, it is possible to measure the booting energy consumption and from there, guess with great precision even private keys. This is not magic – it is Science! Physics ftw \m/ (btw, remember about the “ideal conditions” you learn at school – they will never happen!)

Data flow in the PCB

What is the point of having data if you can not read, write or even transfer it someway? Having a good look at your board should be enough to give you an idea how the data flows through it. You can take the IC as a starting point and follow the PCB tracks to create your own map. Another important information: most of the microcontrollers have a

datasheet living on the Internet, so don't be shy and go duck duck it. They are publicly available and have all the technical information needed for the beginning - stuff like pinouts, communication protocols used, the voltage, and so on). It is always worth spending some time having a look at your hardware components and exploring the online catalogue.

> Kylma added a link to some datasheets

Let's start the hands-on now in the budget form:

Accessing the code

Dumping the code from the Integrated Circuit - short **IC** - and unpacking it are the steps described here. This can be the easiest and the most complex part of the process. You don't need any expensive device and there are plenty of freeware tools available (many also open source!).

As I stated before, most of the devices have datasheets available online. Find a datasheet for your IC enables you to skip the process of identifying the pinouts \m/ Now, we need to identify how the communication works on the board. Here we are actively looking for signals which could interfere in the data dumping / reading. How to avoid the data interference during the dumping depends highly on the device you have in your hands.

Do you mean I need to desolder?

This would be the obvious way to go, right? If you don't want other components of the board interfering the signal received from the IC you are analyzing, disconnect it from the board and go go go! This method gives you total control of the IC and it is lots of fun! But… this process requires some experience and TIME. The idea here is to keep it as simple as it gets and as cheap as it gets, so try to avoid this situation and use your "creativity". It is possible to create something to monitor the other devices on the board and supply power just to your IC? This is just one side idea :)

As soon as your IC is isolated, connect it to a power supply and a computer, so that you can start reading the memory, block per block. This part can take some time, so don't think much and go grab a coffee :)

Dumping the code

To create your own tools can be really fun, but it costs time and it is way easier when you actually know what you are doing ;) So if it is your first hardware, I would recommend you to use some nice open source projects available out there.

To dump the flash, I really like flashrom which is indeed one of the most popular tools for it. It is easy to use, a bit buggy but it works in my Rasperry Pi :) file is always useful and I start nothing before at least having an idea about the fie format I am dealing with. So if file doesn't work, you can always go back to the unfamous binwalk.

From the binwalk logs, it is possible to recover important addresses. With them you can use dd to split the binaries into specific segments. The dd command needs some arguments like *bs* (block size), *skip* (offset) and *size*. You are probably going to have at least three segments:

- bootloader.bin: often not encrypted, used directly by the microcontroller so yeah…
- kernel.bin: if you are lucky, it will be some kind of Linux Kernel. This is the code which controls the bare metal :) Often the most fun one to reverse and often compressed. Again, use file to find out the best way to decompress it.
- rootfs.bin: this is the file system containing all the system files, configuration files and also system binaries. Again, use file to find out the best way to decompress it.

If the image is encrypted, decrypt it using fcrackzip.

Cheatsheet

```
$ dd if=image.zip of=firmware.zip bs=64 skip=1 # extract fir
$ fcrackzip file # cracks zip file
$ binwalk -e file.bin # extract compress bin images
$ unyaffs file # extract .unyaffs2 filesystem mount points
```

Post written originally in pt_br as a guest post to Mente Binaria and it wouldn't be here without the precious help of Kylma.

Author: barbie aka Thaís