Trainee Manual

Hackable ESP Device

IoT Cyber Security Course

5 Dollar ESP Controller

ESPinoza - 24/01/2022

Version 0.1

Team:
Thijs Takken
Luke de Munk
Christina Kostine
Twenne Elffers

Hogeschool van Amsterdam

Table of contents

Version history

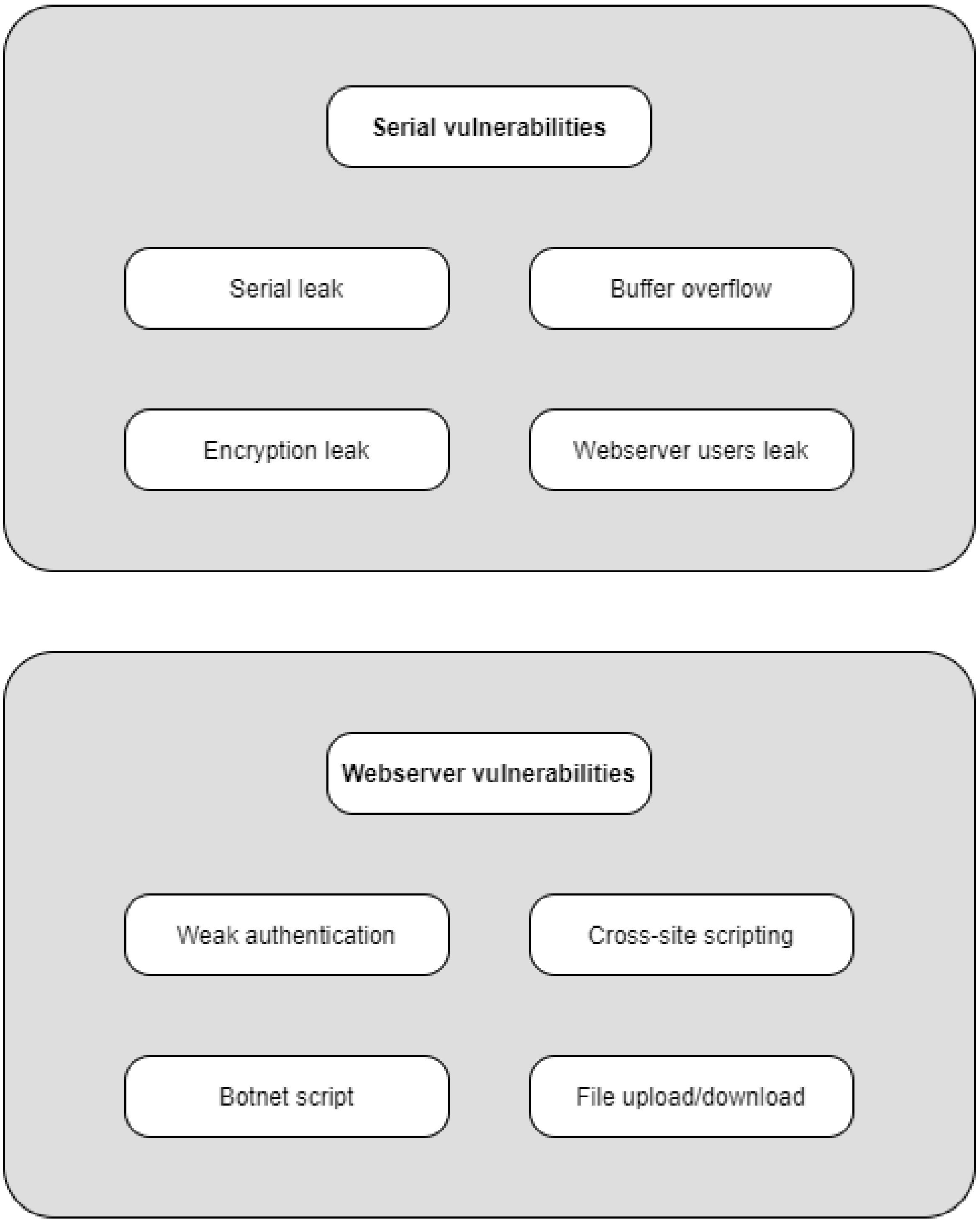
Version	Changes	Date	Author
0.1	Initial version	10-9-2021	Team 1
1.0	Final Draft	31/01/2022	Team 1

1. Introduction

Welcome to the Eurofins IoT Cyber Security Training. This document gives insight into the number of vulnerabilities and challenges. You can use this guide to get started with hacking the device, since it contains descriptions and hints for all the vulnerabilities. However the literal answers are not in this document, it is recommended to first try to find vulnerabilities by just exploring the device. By this way, you can see how big your knowledge already is.

When you are stuck or think you have all the vulnerabilities, you can take a look at this guide to make sure you have everything. The first chapter describes all the vulnerabilities on the device abstract and in the second chapter, you can find hints for each vulnerability.

2. Overview of the vulnerabilities



3. Vulnerabilities

In this chapter all the implemented vulnerabilities are listed, accompanied with a description and hints if needed. The order is based on the OWASP IoT top ten. This way there is a standardized way of ordering and structuring the vulnerabilities.

3.1 Weak, Guessable or Hardcoded Passwords

Use of easily brute forced, publicly available, or unchangeable credentials, including backdoors in firmware or client software that grants unauthorized access to deployed systems.

3.1.1 Serial leak (& web server users leak & encryption leak)

The serial console is often used by testing software on microchips. The debug messages can contain sensitive data. This device has the capability to interpret several commands as well. The serial is in this case not turned off before the deployment, which can lead to serious security risks. Especially when commands can be executed, the data on the device is not secure.

The ‘Exploitable configuration upload & download’ vulnerability is also related to this category.

3.2 Insecure Network services

Unneeded or insecure network services running on the device itself, especially those exposed to the internet, that compromise the confidentiality, integrity/authenticity, or availability of information or allow unauthorized remote control.

3.2.1 Cross-site request forgery

Through the beautiful webpage, users can control their LED controller. But how does this work? Maybe it’s controllable?!

3.2.2 BotLED

Let’s take over all the LED controllers! For this assignment use the provided **BotLED_skel.py** file. Basic knowledge on Python is required.

3.3 Insecure EcoSystem Interfaces

Insecure web, backend API, cloud, or mobile interfaces in the ecosystem outside of the device that allows compromise of the device or its related components. Common issues include a lack of authentication/authorization, lacking or weak encryption, and a lack of input and output filtering.

3.3.1 Encryption Leak

CBC mode is an AES block cipher mode that is XOR-ing the first plaintext block with an initialization vector before encrypting it. The decryption works in the same way with ciphered text.

Even if Cbc mode is more secure than EBC one as it hides the patterns of plaintext. This vulnerability is about finding the key and decrypting the file with it.

3.3.2 Buffer overflow

In the past, buffer overflows were common. Today, most of the software is written in a way that a buffer overflow is not possible anymore. In microchips such as the ESP device, buffer overflows can still exist. A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the storage capacity of the memory buffer. As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations.

For example, a buffer for log-in credentials may be designed to expect username and password inputs of 8 bytes, so if a transaction involves an input of 10 bytes (that is, 2 bytes more than expected), the program may write the excess data past the buffer boundary.

If the transaction overwrites executable code, it can cause the program to behave unpredictably and generate incorrect results, memory access errors, or crashes. If attackers know the memory layout of a program, they can intentionally feed input that the buffer cannot store, and overwrite areas that hold executable code, replacing it with their own code. For example, an attacker can overwrite the program-pointer (an object that points to another area in memory) and point it to an exploit payload, to gain control over the program.

3.4 Lack of Secure Update Mechanism

Lack of ability to securely update the device. This includes lack of firmware validation on devices, lack of secure delivery (un-encrypted in transit), lack of anti-rollback mechanisms, and lack of notifications of security changes due to updates.

3.4.1 Exploitable configuration upload & download

There is a configuration file hidden somewhere on the web server. This configuration can be back-uped an restored. The restoring and back-uping has not been properly secured. Try exploiting it.

3.5 Use of Insecure or Outdated Components

Use of deprecated or insecure software components/libraries that could allow the device to be compromised. This includes insecure customization of operating system platforms, and the use of third-party software or hardware components from a compromised supply chain

We bet that there are insecure components that can be hardware hacked, but this was not part of the current project.

3.6 Insufficient Privacy Protection

User's personal information stored on the device or in the ecosystem that is used insecurely, improperly, or without permission.

The following vulnerabilities have implications on the privacy category.

1. Serial leak
2. Web server users leak
3. Exploitable configuration upload & download

3.7 Insecure Data Transfer and Storage

Lack of encryption or access control of sensitive data anywhere within the ecosystem, including at rest, in transit, or during processing

The 'Exploitable configuration upload & download' vulnerability is also related to this category.

3.8 Lack of Device Management

Lack of security support on devices deployed in production, including asset management, update management, secure decommissioning, systems monitoring, and response capabilities.

There is no remote device management present, therefore no vulnerabilities in this category are present.

3.9 Insecure Default Settings

Devices or systems shipped with insecure default settings or lack the ability to make the system more secure by restricting operators from modifying configurations.

The following vulnerabilities are using some default and hard coded variables, therefore they are shipped with the same hard coded variables every time.

1. Serial leak
2. Web server users leak
3. Encryption leak
4. Exploitable configuration upload & download

3.10 Lack of Physical Hardening

Lack of physical hardening measures, allowing potential attackers to gain sensitive information that can help in a future remote attack or take local control of the device.

This category is broken by design, every trainee receives a device and physical hardening is therefore always broken by design.

4. Hints

IMPORTANT! On the next pages you can find hints. When you have not explored the device yet, it is recommended that you do that first!

4.1 Serial vulnerabilities

The list below tells how to get started with the serial vulnerabilities.

1. Connect the device via the USB or the GPIO pins
2. Research common baud rates and try some
3. Research the possible commands

4.1.1 Serial leak

1. Enable the full debug mode by a command
2. Reboot the device
3. Look into the startup bytes

4.1.2 Web server users leak

1. Become a superuser in one of the two ways
2. Enter a command to list all the web server users

4.1.3 Encryption leak

CBC mode is an AES block cipher mode that is XOR-ing the first plaintext block with an initialization vector before encrypting it. The decryption works in the same way with ciphered text.
Even if Cbc mode is more secure than EBC one as it hides the patterns of plaintext. This vulnerability is about finding the key and decrypting the file with it.

1. Become a superuser in one of the two ways
2. Look at the commands

4.1.4 Buffer overflow

1. Look at the possible commands
2. Look into the files with a command
3. The program pointer needs to be overwritten by the overflow
4. Important lines in program file:
    a. 17, 19
5. Important line in disassembled program file
    a. 1, 27
6. Try to run the program with arguments, it takes some tries
7. The way of inputting hexadecimals is, for example: “\x11”

4.2 Web server vulnerabilities

4.2.1 Exploitable configuration upload & download

There is a configuration file which is used for the webserver. There are three actions that can be executed to exploit two information pieces.

1. Try the “user” page
2. Have you looked into common weak passwords?

1. Have a look at “config.conf”
2. Look for the admin password
3. Maybe try looking in the serial interface for a way to decrypt the file?
4. gpg –decrypt key string is the correct command
5. The password is located between the double dots

1. There is an admin page with an extra feature
2. Try uploading your own file?

4.2.2 Cross-site request forgery

The main webpage has a function to set the power state. This controls what the LED does.

1. Inspect the set_power function
2. How do you interact with it? Use 0 or 1
3. Create an html document in which you call the right url, this way, whenever a victim uses your webpage, in theory, you as the hacker can control their LED controller

4.2.3 BotLED

The LED controller has a callable function with which you can control the LED. In this challenge it is the goal to control a whole group of ESP's on a network and manipulate their LED state.

1. Use the URL Library (urllib)
2. Use it to do a request with urlopen
3. Leverage the variables "state" and "addr" in the url to call the devices with
4. Remember, the code already does the loop, you only have to worry about making a single call, no other logic required