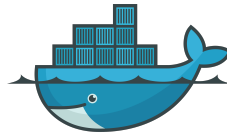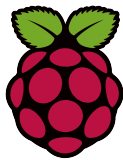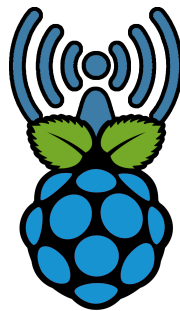# PiTest

Penetration Testing on a mobile platform
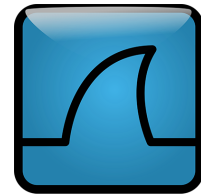
University of Padua

Project documentation

filippo.berto.4@studenti.unipd.it, berto.f@protonmail.com

Version	1.0.0
Redaction date	2017-06-30
Distribution	*Prof. Armir Bujari*

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduction

1.1 Purpose of the document

This document describes the project, the analysis of the problem, its design, the choices taken, its use and how to extend its functionality.

1.2 Purpose of the product

This product has the purpose of simplify the use of security research software on a mobile context, allowing the user to access a complete set of tools from an easy to use interface on a mobile phone.
The final product is going to offer the following functionality:

- Secure login through a REST API;

- Obtain the list of allowed commands;

- Execute common tasks, as network and WiFi scanning;

- Execute predefined scripts;

- Execute arbitrary commands, give by the user;

- Connect to the device through a WiFi hot-spot;

- Control the device using an Android App client;

- Show the results of the commands to the user in a simple GUI.

1.3 Useful references

- **Project presentation**: www.google.com/drive;

- **GitLab documentation**: www.gitlab.com;

- **Git documentation**: www.git.org

- **Android documentation**: developers.android.com

- **NodeJS documentation**: TODO

- **Docker documentation**: TODO

- **Arch Linux Wiki**: wiki.archlinux.org

- **Black Arch documentation**: TODO

- **Raspberry Pi documentation**: TODO

- **Aircrack-ng documentation**: TODO

- **Wireshark documentation**: TODO

2 Problem analysis

2.1 Problematic factors

The activity of penetration testing, or security auditing, pose difficulties in various in both hardware and software fields, particularly the following ones:

Penetration testing tools Most penetration testing tools only work on a Linux environment and some may require dependencies which can conflict with each other.

Penetration testing in mobility Some tasks need to be executed on the field, and carrying a laptop could be difficult in some situation. A small and portable device is necessary to accomplish the work.

Penetration testing and computational power The common tasks of security auditing can be split in two genres: the ones which need lot of computational power, such as hash cracking, and the ones which need less, such as WiFi auditing and network traffic sniffing. Since the platform chosen is mobile, its computational power and it battery life are limited, so the project will focus mostly on the second ones.

Penetration testing hardware Most tasks need some specific hardware to be executable, such as Software Defined Radios (SDR) or WiFi adapters. The device must have the general connectivity needed to allow the use of those tools.

2.2 Solution proposed

The following solution has been chosen as the most adequate to mitigate or completely solve all the problems mentioned:

Hardware The platform chosen for the project is a Rapsberry Pi B+, a System On a Chip board, featuring a single core ARM v6 processor, 512MB of RAM memory, 4 USB ports and a Ethernet adapter, plus 40 GPIO expansion pins for extra expansion. The board is powerful enough to accomplish common tasks and its power consumption is low enough to be successfully powered my a USB power bank. Also the board costs about 35$, meaning the total cost is very contained.

Penetration testing hardware For WiFi analysis I choose to use an ALFA AWS051NH v2 adapter, capable of both monitor mode and packet injection, necessary in most WiFi attacks.

Operating system The operating system is Arch Linux a lightweight GNU Linux distribution that is easy to modify for the user needs.

Architecture The architecture chosen is a simple server-client, since the network used to communicate between the Raspberry Pi and the client is self contained and most of the time the server will have only one client connected at a time.

network   Since the necessity of a simple way to connect to the board and transfer any kind of data between the server and the client, I have choose to generate a WiFi hot-spot on the board; this way the connection is wireless and fast and the Ethernet port is free to use. The 3 USB ports remaining can still be used to control directly the board or to plug in other devices.

Software   The main difficulty with software is compatibility, so the most effective solution is to virtualize a minified system with only the applications needed for the task. The most effective way to achieve this is using a Docker container, specifically built for that single application. This solution allows controlled transparency of both hardware and software, excluding the possibility of dependency collision and incompatibility.

As base to the container software I have chosen to use a minified version of Black Arch, an Arch Linux based distribution with built in support for a variety of penetration testing software, still maintaining small size and ease of use.

The main application, which exposes a REST API and allows to execute given commands, scripts and tasks is a NodeJS server. This choice allows a simple development, given the modularity of the framework, and cross-platform compatibility.

The network is controlled by a hostapd daemon on the main operating system and the network is managed by both a dhcpcd and a dnsmasq services, to allow network NAT.

3 Project definition

The project definition will be split in two sections, describing the server and on the client.

3.1 Server

The server application, from now on *PiTest*, is installed on the Operating System of the board and needs NodeJS to work.

3.1.1 Architecture diagram

The following is an high level diagram of the architecture of *PiTest*.
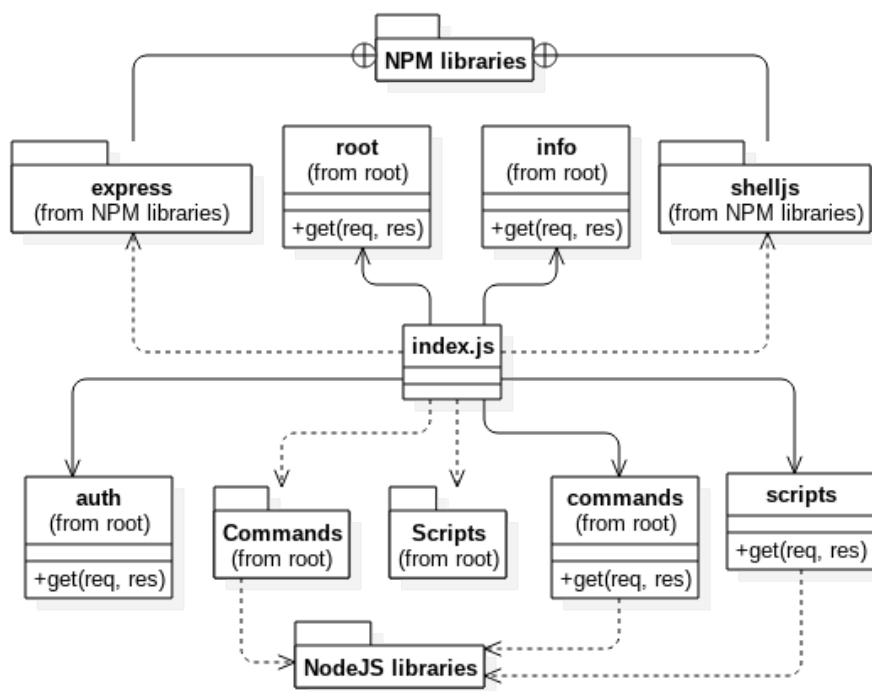
Figura 1: High level architectural diagram

3.1.2 Low level class diagram

Figura 2: Low level UML diagram of PiTest

3.1.3 Modules

3.1.3.1 index.js This is the main file of the application, it creates an Express app and loads all the routes of the web app.

3.1.3.2 root Defines the behaviour of the root page (`/`), it's purpose is to respond with a custom welcome message.

3.1.3.3 info Defines the behaviour of the root `/info`, it gives information about the server, as its name, version and author; it requires authentication using a valid token.
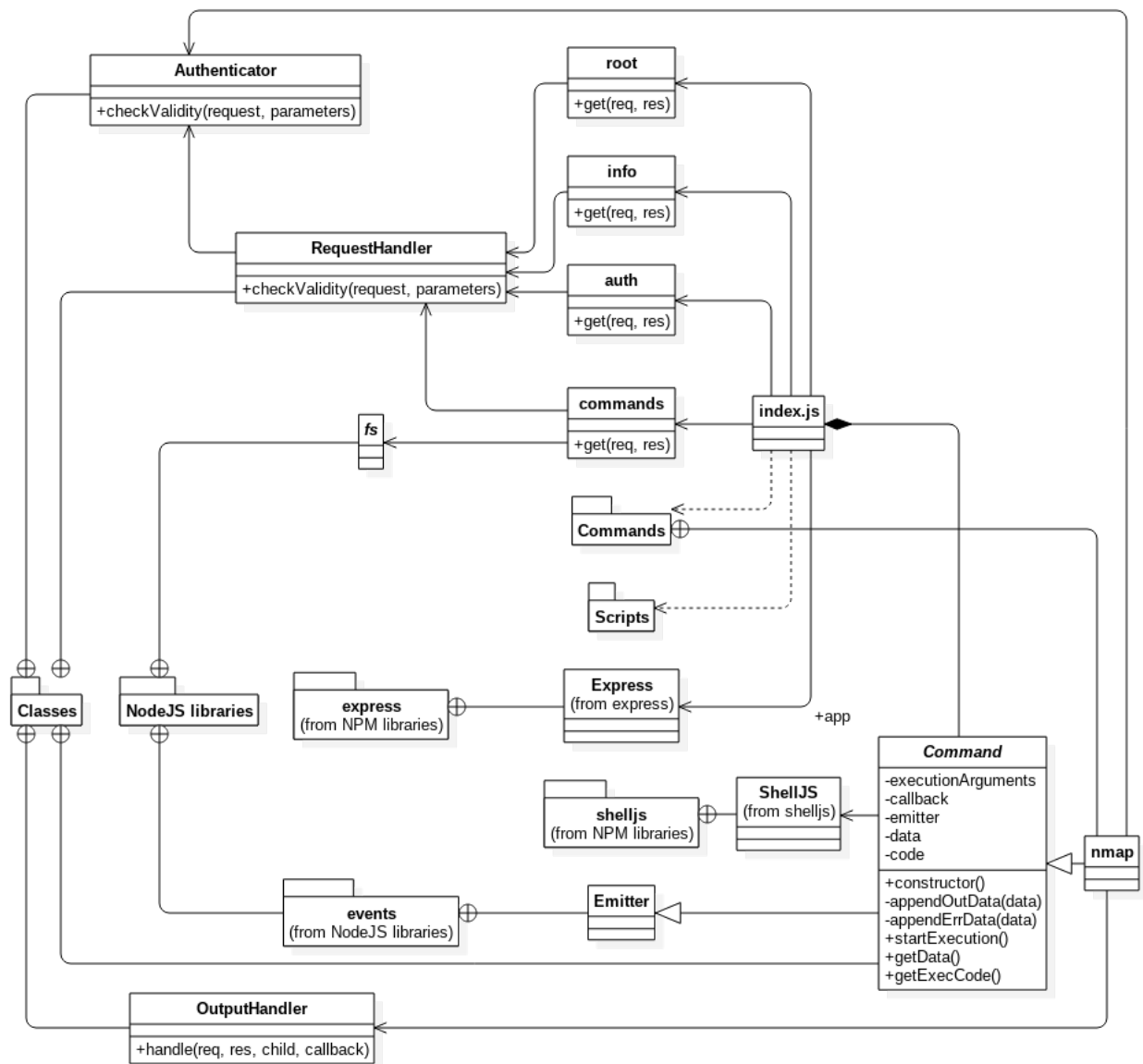
3.1.3.4 auth Defines the behaviour of the root `/auth`, it's used to authenticate the user and requires a valid token.

3.1.3.5 commands Defines the behaviour of the root `/commands`, it's used to list to the user the possible commands and all their parameters; it requires authentication using a valid token.

3.1.3.6 scripts Defines GTE behaviour of the root `/scripts`, it's used to list to the user the possible scripts and all their parameters; it requires authentication using a valid token.

3.1.4 Dependencies

Since the necessity of a solid code base from which extend the application, I have chosen to use some common libraries, both from the standard NodeJS ones and from NPM modules.

3.1.4.1 fs It's a NodeJS standard library which allows to read files on the file system.

3.1.4.2 events It's a NodeJS standard library which allows to use the event system.

3.1.4.3 express It's a *NPM* library which allows to generate a complete web server; it's used as a base for the REST API. (https://www.npmjs.com/package/express)

3.1.4.4 shelljs It's a *NPM* library which allows to execute shell commands on the host in bot synchronous and asynchronous ways. It's used to call the host executables from the server.

3.1.5 Commands and scripts

To easily control the execution of commands and scripts I have chosen to create two respective wrapper classes, which contain all the information needed for the execution, have a simple interface to start the execution and retrieve the result with a callback. Also, every time there is an output, both on `STDERR` or `STDOUT` from the shell, an event, containing the relevant data, is created.

3.1.6   Virtualization for abstraction

Since not all applications can coexist on the same Operating System I have chosen to use Docker to separate and abstract the layer of control from the layer of execution. This way, it is possible to work with several distributions, in parallel, on the same system, and each one, possibly, with a different configuration. Every preconfigured command works on a Docker container, so the host is kept safe from possible vulnerabilities, while maintaining all its potential.

3.2   Client

<Descrizione> TODO

3.2.1   Interface

The interface of the client will allow the user to test the connection to the server with an authentication test. Next, if the connection is correct, the app will show a list of possible commands, which contents depend on the commands allowed by the server. Each of this command will have a short description and an execute button. When the command is executed the app will show the result of the execution in a customized graphic, if possible. It is also possible to execute an arbitrary command: the user is allowed to enter the full command in a text box.

3.2.2   REST CALLS

Every communication ongoing between the client and the server is through a REST API. This way the client can manage multiple asynchronous commands in parallel and maintain an high level of security, as every call needs to be authenticated by a token.

3.2.3   Command output customization

The use of a GUI allow to show the output of the server with a simple interface, which can be less intimidating than a shell for a non experienced user.