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**: `google.com/drive`;
- **GitLab documentation**: `docs.gitlab.com`;
- **Git documentation**: `git-scm.org/documentation`;
- **Android documentation**: `developers.android.com`;
- **NodeJS documentation**: `nodejs.org/docs`;
- **Docker documentation**: `docs.socker.com`;
- **Arch Linux Wiki**: `wiki.archlinux.org`;
- **Black Arch documentation**: `blackarch.org/guide`;
- **Raspberry Pi documentation**: `raspberrypi.org/documentation`;
- **Aircrack-ng documentation**: `aircrack-ng.org/documentation`;
- **Wireshark documentation**: `wireshark.org/docs`.

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*.

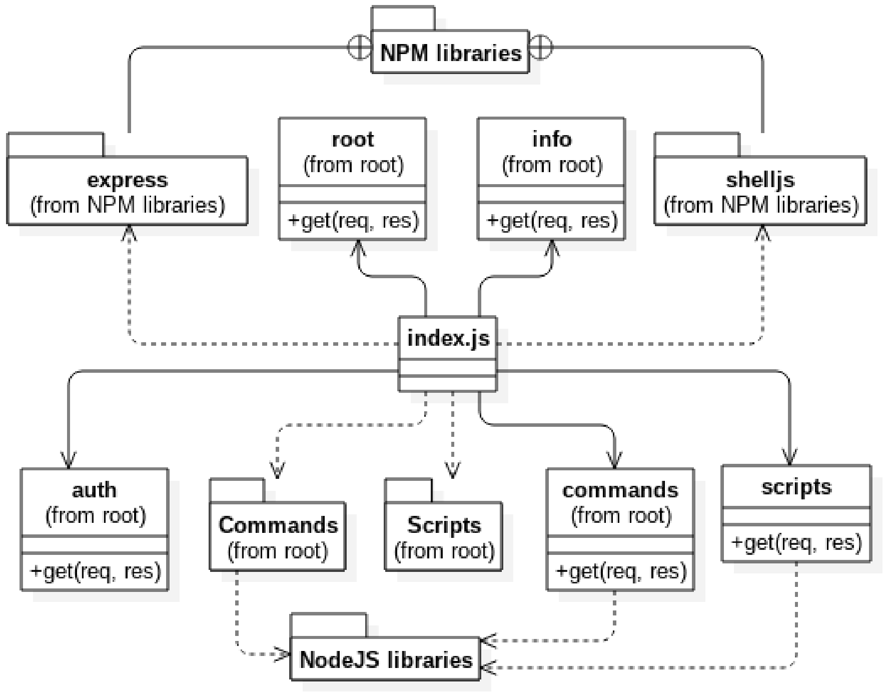

Figura 1: High level architectural diagram

3.1.2 Low level class diagram

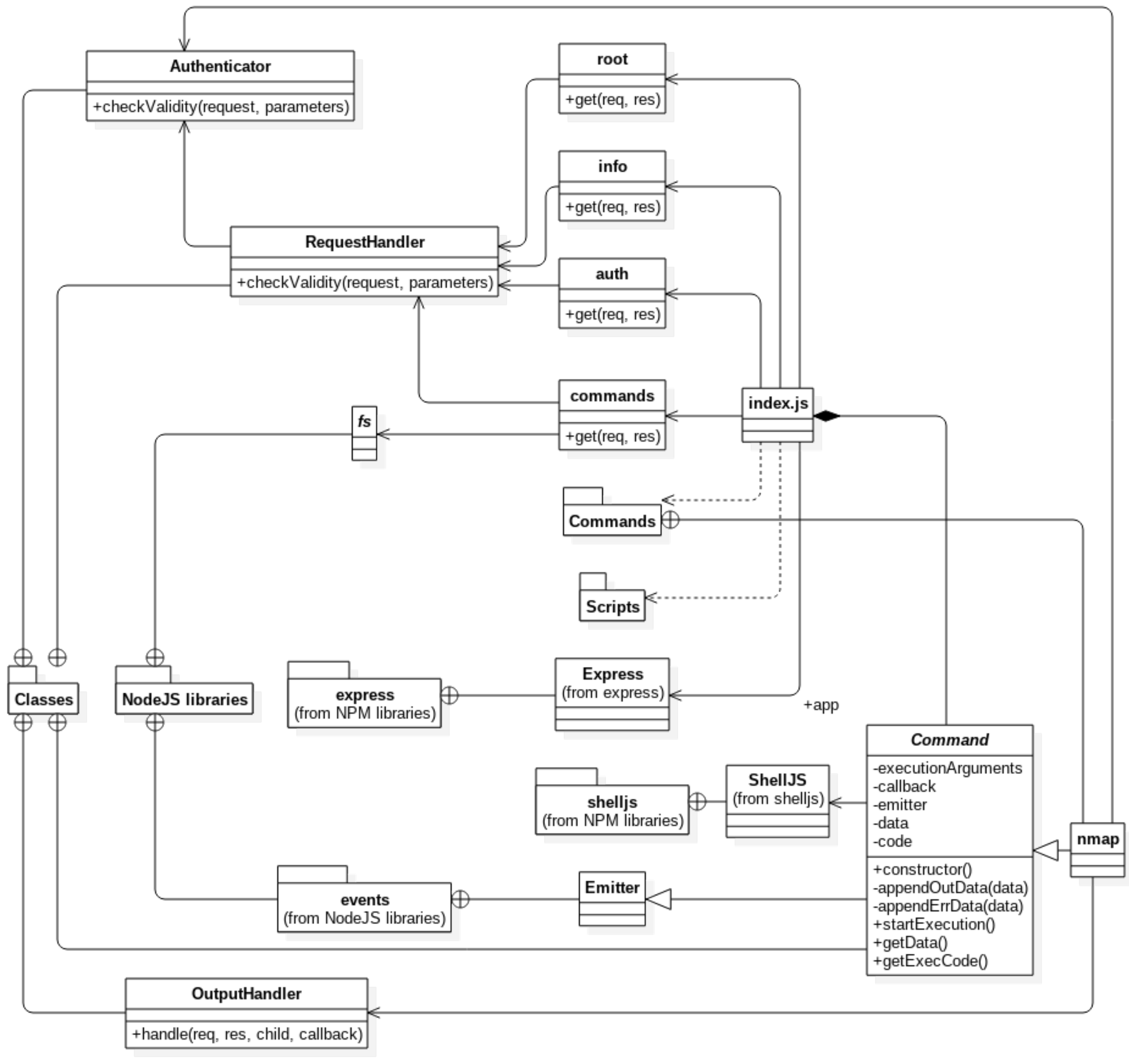

Figura 2: Low level UML diagram of PiTest

3.1.3 Modules

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

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

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.

auth Defines the behaviour of the root `/auth`, it is used to authenticate the user and requires a valid token.

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

scripts Defines GTE behaviour of the root `/scripts`, it is 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.

fs It is a *NodeJS*standard library which allows to read files on the file system.

events It is a *NodeJS*standard library which allows to use the event system.

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

shelljs It is a *NPM*library which allows to execute shell commands on the host in bot synchronous and asynchronous ways. It is 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

The client side of the project is an Android application. I have chosen to use android as platform because it is the most common platform for mobile phones and ease access to all the libraries needed to do simple *REST*calls. I have chosen this platform also because it is new to me, so I can learn something new with this project.

3.2.1 Dependencies

Apache HTTP client Used to manage the *REST*calls and retrieve the information from the server.

3.2.2 Interface

The interface of the client will allow the user to setup the connection to the server with an authentication test. Next, if the connection is correct, the main menu, allowing the user to execute an arbitrary command, execute a preconfigured command or execute a preconfigured script. If the user choose to execute an arbitrary command, he will be prompted with a text box to enter it and an execute button. After the execution is completed, a text view will be populated with the text result. If the user choose to execute a preconfigured script or command, he will be prompted with a list of possible choices, each one with a simple description. Selecting an entry will start the execution; after which completion a text view will be populated with the result. Some preconfigured commands and scripts can have a custom view, to better show the results achieved.

3.2.3 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. I have chosen to use *REST*as it is more battery efficient than his common alternative, using a websocket, since it does not keep the connection open, sending ping-pong packets. Also, *REST*is very simple to port on other platforms, allowing further expansion of the project.

3.2.4 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. An example would be a *airodump-ng* scan, better presented with a list of items, with each own description.

3.3 System configuration

The system will work on two machines, each with its own configuration, as explained in the next sections.

3.3.1 Server

The server application can generally run on any platform which supports *NodeJS*8 and *Docker*, but the device which it will be deployed on is a Raspberry Pi B+.

3.3.1.1 Network The system requires at least a WiFi adapter, to generate the common network between the Raspberry and the smartphone, and another one to execute WiFi networks analysis. The network architecture chosen is a station-client one and the Raspberry is the host, meaning it needs a hostapd service to control the WiFi adapter, a dhcpcd one to route dynamically the client and a dnsmasq one to route the traffic. The Raspberry will act as a WiFi router, using NAT on its clients.
The complete guide to setup the network tools on a Arch Linux system can be found in the server folder on the repository.

3.3.1.2 Installation and configuration To run the server application *Docker*and *NodeJS*and its package manager *NPM*must be installed on the host.
Download the latest version of PiTest form the repository using a browser or *git*.
Build a *Docker*container to contain the application you want to run on the system; a *Black Arch* dockerfile example is present in the server folder on the repository. It is possible to build as many container as needed to run all the applications wanted.
Install all the packages necessary using `npm install` inside the node server folder; after the operation has completed you can start the server with the command `npm start`.
In case you want to run all the tests of the server you can do it by using the command `npm test` inside the above mentioned folder.
It is good practice to create a service file to automate the execution of the server on the system boot using systemd; the procedure may vary according to the Linux distribution chosen.

3.3.1.3 Security All the *REST*calls, except the ping, need a token parameter to authenticate the user. The token is unique in the system and can be configured in the config.json file in the server folder. Also the port used to connect to the server can be changed in the same file.

3.3.2 Client

3.3.2.1 Build The build process of the client is standard and automated thanks to *Gradle*. A default *Android Studio* installation will be sufficient to build the application.

3.3.2.2 Setup The client needs an initial setup procedure to connect to the server; during which it will the user will be asked for the address of the server, the port used to connect to the server and the token for the authentication.