Project

Design Laboratory

A device for tracking network traffic

Authors: Mateusz Furgała, Maciej Dawczak

Elektronika i Telekomunikacja, 3 rok

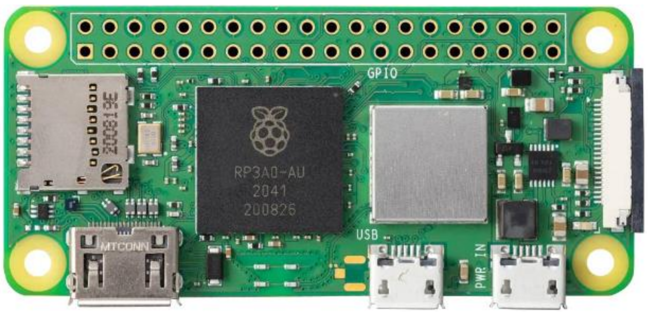
Date: 24.01.2024

1) Introduction

The goal of the project was to build a device that could track network traffic. It would function as an access point, providing Wi-Fi to users who could connect to the hotspot. The device would automatically record the network traffic, which could then be checked and reviewed later.

2) Specyfikacja techniczna

a. Rasbberry PI Zero 2 W



The Raspberry Pi Zero 2 W is a compact single-board computer (SBC) developed by the Raspberry Pi Foundation, offering good performance in a small size.

- **Processor:**

Equipped with a quad-core ARM Cortex-A53 processor running at 1 GHz.

- **Memory:**

512 MB of RAM.

- **Wireless Communication:**

Integrated WiFi module (2.4 GHz).
Support for Bluetooth 4.2.

- **Connectors and Interfaces:**

GPIO (General Purpose Input/Output) connector.
Micro USB for power and an HDMI port for video output.

- **Size and Form:**

Small dimensions (65 mm x 30 mm).

- **Operating System:**

Compatible with various operating systems, including the official Raspbian system, tailored for ARM architecture.

b. Network card

The T3U Nano is a wireless network adapter by TP-Link, its features include:

- Wireless connections with speeds up to 1300 Mb/s (400 Mb/s in the 2.4 GHz band and 867 Mb/s in the 5 GHz band).

- Support for both 2.4 GHz and 5 GHz bands allows for the use of the speed and range of the latest dual-band routers.

c. Power delivery

The power supply used is the LiPo SHIM, an add-on for the Raspberry Pi that mounts directly onto the GPIO pins of the mini-computer. The device has a JST 2-pin connector for connecting a Li-Pol or Li-Ion battery to power the Raspberry Pi. A TPS61232 step-up converter is used on the board. The module features LED indicators for power status and low battery level. Using online calculator we estimated that our device should run about 13-14 hour on one charge.

- Board Thickness: 0.8 mm
- JST 2.0 mm 2-pin connector
- Battery charge indicator via LED light
- Provides a constant current of up to 1.5 A (15 uA in idle state)
- Low battery warning at 3.4 V
- Automatic shutdown at 3 V to protect the battery
- Terminals for VBAT+, GND, and EN
- Battery capacity is 1200mAh.

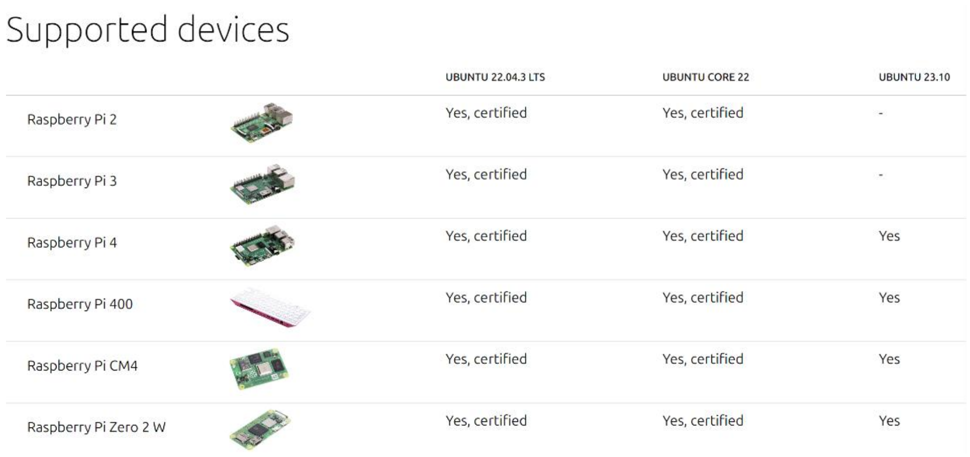
d. Device case

For assembling the device, a specially designed and 3D-printed enclosure will be used. It should accommodate all the components as well as the battery.

e. Operating system

Ubuntu

As the operating system, Linux Ubuntu 22.04.3 LTS was chosen for its lightweight nature and compatibility certification.

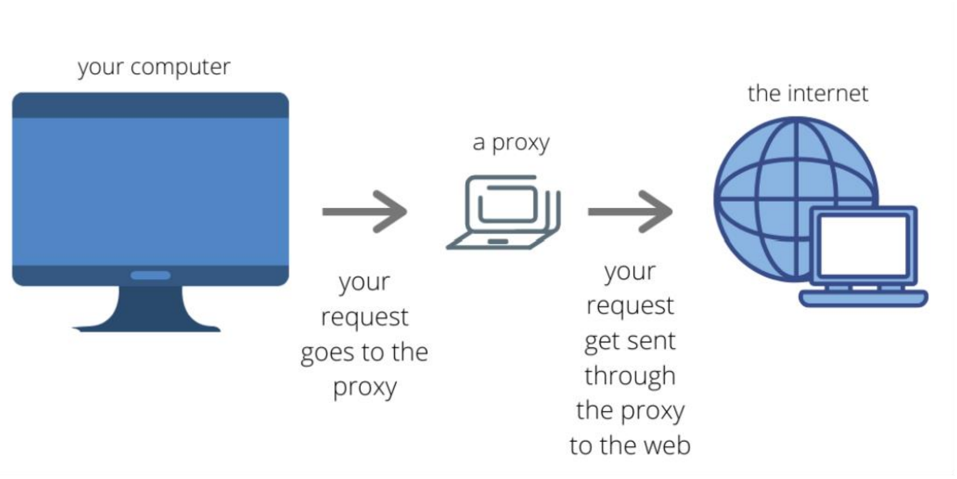


Supported devices

		UBUNTU 22.04.3 LTS	UBUNTU CORE 22	UBUNTU 23.10
Raspberry Pi 2		Yes, certified	Yes, certified	-
Raspberry Pi 3		Yes, certified	Yes, certified	-
Raspberry Pi 4		Yes, certified	Yes, certified	Yes
Raspberry Pi 400		Yes, certified	Yes, certified	Yes
Raspberry Pi CM4		Yes, certified	Yes, certified	Yes
Raspberry Pi Zero 2 W		Yes, certified	Yes, certified	Yes

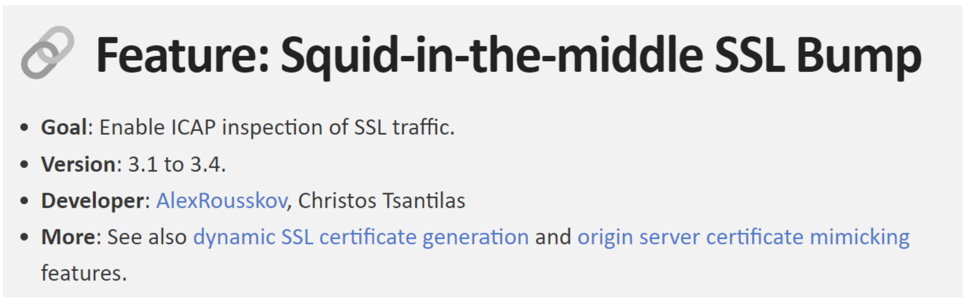
3) <u>Configuration of Squid proxy</u>

To capture encrypted network traffic, a Squid proxy server was configured on the device. By design, a proxy server acts as a kind of tunnel between the client and the server and is not capable of "peeking" into encrypted network traffic.


your computer
your request goes to the proxy
a proxy
your request get sent through the proxy to the web
the internet

However, it is possible to configure a proxy server in such a way that it can read packets secured by the TLS protocol. Our proxy acts as a so-called "Man in the Middle," where client requests are received by the server, which then initiates a connection with the target server. This allows us to establish a connection using the TLS 1.0/1.1 protocol on one side, and the newer and more secure TLS 1.2 protocol required by most servers on the other.

The newer version of the TLS protocol (1.2) requires that certificates be "signed" by a so-called CA (Certified Authority), whereas in TLS 1.0, certificates can be "self-signed." However, it is still important that such a certificate is trusted by the client. We will use the ssl_bump function for this purpose.



Feature: Squid-in-the-middle SSL Bump

- **Goal**: Enable ICAP inspection of SSL traffic.
- **Version**: 3.1 to 3.4.
- **Developer**: AlexRousskov, Christos Tsantilas
- **More**: See also dynamic SSL certificate generation and origin server certificate mimicking features.

SSL Bump Functions Peek and Splice:

ssl_bump allows the differentiation of SSL/TLS traffic into two main categories: peek and splice.

- Peek: The peek mechanism allows for a "peek" into the traffic without interfering with the transmission itself. This enables administrators to analyse data without needing to decode it.
- Splice: The splice mechanism allows for the decoding of SSL/TLS traffic and intervention in the transmission, enabling monitoring and/or modification of the transmitted data.

Access Control:

- ssl_bump enables control over access to resources secured by the SSL/TLS protocol. Rules can be defined based on various criteria, such as URLs, content types, or session characteristics.

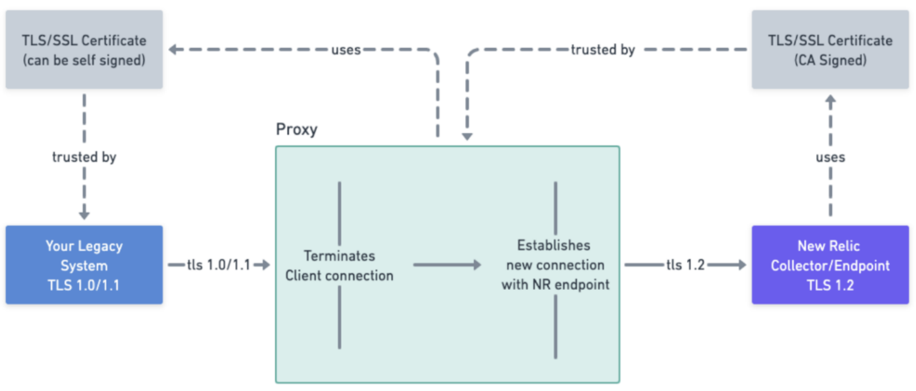
Certificates:

- ssl_bump allows for the management of SSL/TLS certificates. The proxy can generate its own certificates in response to requests from the client and/or server, facilitating the configuration of SSL/TLS at the proxy level.

Traffic Recognition:

- ssl_bump enables the proxy to effectively recognize the type of SSL/TLS traffic, allowing for more precise control and analysis.

Squid proxy setup


TLS/SSL Certificate (can be self signed)
uses
trusted by
TLS/SSL Certificate (CA Signed)
Proxy
trusted by
uses
Your Legacy System TLS 1.0/1.1
tls 1.0/1.1
Terminates Client connection
Establishes new connection with NR endpoint
tls 1.2
New Relic Collector/Endpoint TLS 1.2

Configuration on Linux:

```
maciej@kupa:~$ sudo apt-get update

maciej@kupa:~$ sudo apt-get install -y \ squid-common \ squid \
squidclient
```

Squid server download

```
maciej@kupa:~$ sudo /usr/lib/squid/security_file_certgen -c -s
/var/spool/squid_ssldb -M 4MB
```

Initialization of the certificate database

```
maciej@kupa:~$ sudo /usr/sbin/squid -YC --foreground -f
/opt/conf/squid/squid.conf
```

Starting the server as a foreground process

Configuring and modifying the operation of the server is possible by editing the sqid.conf file which can be downloaded from github.

```
maciej@kupa:~$
git clone https://github.com/gmellini/squidproxy-conf.git

maciej@kupa:~$ docker run  -p 3128:3128 -ti docker.io/salrashid123/squidproxy
                /apps/squid/sbin/squid -NsY -f /apps/squid.conf.intercept
```

Software

4) Analiza rynku

The main and most recognizable device on the market is Flipper Zero

Price: about 1000 PLN

Flipper Zero is a multifunctional device for security testing, system exploration, and remote control of various devices. Its design and software are publicly available, allowing the community to co-create, analyse code, and develop new features.

Features:

- **Security and Hacking:**

Flipper Zero has features related to security testing, including packet capture, attacks on wireless security, penetration tests, etc.

- **Remote Control:**

Capable of remotely controlling various devices, such as TVs, audio systems, and even remote access cards.

- **System Exploration:**

Tools for exploring and studying systems, including network traffic analysis, security testing, etc.

- **Interactivity:**

Equipped with an interactive touch screen, buttons, sensors, and other elements, enabling the use of various functions.

- **Open Source Software:**

Flipper Zero's software is open source, encouraging community collaboration and the development of new features.

- **Multifunctionality:**

Includes a variety of features such as reading RFID cards, testing wireless system security, remote control of devices, etc.

- **Programmability:**

The ability to program and customize the device for different applications.

Another known device is Hackrf

Price: about 3200 PLN

HackRF is a portable tool for hobbyists, researchers, and professionals interested in radio communication and signal processing. It's particularly popular in areas such as signal analysis, wireless security research, and educational purposes in radio communication and electronics.

Our device:

Price: about 500 PLN

Our device is based on the Raspberry Pi Zero 2 W board, so the cost is not high compared to more expensive, specialized devices. It offers many functions because an operating system can be installed on it, which would greatly facilitate the expansion of the device with new features.

5) Functionalities

6) Code description

The packet_saver.cpp program will be responsible for capturing and logging network packets transmitted by specific devices in a local network. Before proceeding with the compilation, it is necessary to configure the access point function on our machine, currently, the built-in Wi-Fi hotspot feature in the Ubuntu system is used for this purpose. Another method of creating a hotspot can be used for this.

Compilation:

Before proceeding with the compilation, it is necessary to check the correctness of the entered name of the interface on which the hotspot will be established.

The program can be compiled using the standard C++ compiler with the following command:

```
g++ packet_saver.cpp -o packet_saver -lpcap
```

Then, the program can be launched in the terminal using command below.

```
sudo ./packet_saver
```

Program Operations:

- Launching the Program: Upon launch, the program enters its main loop, where it cyclically performs its functions.
- Detecting Devices: The program uses arp-scan to identify active devices on the network and creates a list of their IP addresses.
- Starting Packet Capture: For each newly detected IP address, the program initiates a new thread that begins capturing packets.
- Packet Logging: Captured packets are saved to .pcap files, allowing for later analysis.
- Stopping Packet Capture: When a device leaves the network, the capture thread for that device is closed.
- Rechecking Devices: At regular intervals, the program rescans the network to update the list of active devices.

Program Function Descriptions:

I. **string getCurrentDateTime()**
   - Purpose: Generates and returns the current date and time in the format "YYYY-MM-DD_HH-MM-SS".
   - Operation: Uses standard C++ functions such as time, localtime, and ostringstream to format and return the date and time.

II. **void packetHandler(u_char *userData, const struct pcap_pkthdr* pkthdr, const u_char* packet)**
   - Purpose: A callback function used by pcap_dispatch to process captured packets.
   - Operation: Writes the captured packets to a file using pcap_dump. The parameters userData, pkthdr, and packet are used to access packet data and information.

III. **void capturePackets(const string& ip)**
   - Purpose: Captures network packets for a specified IP address.
   - Operation: Configures and opens a pcap session on the selected network interface, sets a filter for the selected IP address, and then captures packets until the activity flag is set to false. Captured packets are saved to a .pcap file.

IV. **void startCapture(const string& ip)**
   - Purpose: Begins the packet capturing process for a given IP address.

- Operation: Sets the activity flag for the given IP address to true and launches a thread with the function capturePackets, passing the IP address as an argument.

V. **void stopCapture(const string& ip)**
   - Purpose: Stops the packet capturing process for a given IP address.
   - Operation: Checks if an active capturing process exists for the given IP address and sets its activity flag to false.

VI. **vector<string> getConnectedIPs()**
   - Purpose: Retrieves a list of active IP addresses in the local network.
   - Operation: Executes the arp-scan system command using popen to find active devices on the network, then reads and returns their IP addresses.

VII. **int main()**
   - Purpose: The main loop of the program, coordinating the capturing process.
   - Operation: In a loop, the program regularly refreshes the list of active IP addresses, initiates packet capturing for newly detected IP addresses, and stops capturing for IP addresses that are no longer active. Uses a mutex to synchronize access to shared resources.