A Framework for Improved Home Network Security

António Craveiro, Ana Oliveira, Jorge Proença, Tiago Cruz, Paulo Simões
Department of Informatics Engineering, University of Coimbra, Portugal
{aamcraveiro, asofiabo}@gmail.com, {jdgomes, tjcruz, psimoes}@dei.uc.pt

Abstract: modern home networks constitute a diverse ecosystem of devices and services, whose management is mostly handled by means of specific service or device provider mechanisms, with only a minority of customers possessing the technical skills required to deal with such tasks. This means that, when it comes to security, most users often exclusively rely on endpoint protection mechanisms (such as anti-virus) to handle their basic needs.

This is mostly due to a basic assumption that considers the home LAN to be a safe environment, with most threats coming from the outside – a premise that underlies most of the research about the subject. However, this perspective predates the widespread adoption of mobile appliances, smart devices and related services, some of which may be abused and/or compromised to spy and exfiltrate sensitive information from their owners. Overall, one cannot consider the home LAN to be a safe environment anymore, as potential vulnerabilities and information leaks may come from within.

This paper addresses the initial phase of the development plan for a security appliance for Home LANs, capable of providing seamless protection for the ever-growing ecosystem of devices and services which are common on such networks, while also addressing the specific requirements imposed by multi-tenancy and cohabitation with third-party services. By monitoring and controlling network traffic flows on a per-device basis, we intend to make it possible to characterize typical traffic profiles for each type of device, to detect anomalies and/or information leaking events. Eventually, this capability may be leveraged to provide to control device-to-device communications in order to block unwanted interactions between equipment and external sites. By moving beyond the last mile and into the customers' doorsteps, the proposed solution intends to provide comprehensive and contextual security monitoring on a per-user basis, embedded at the residential gateway or eventually resorting to a specialized appliance deployed in the home LAN.

Keywords: Home Network Security, Distributed Security, Smart Homes, Home appliances.

1. Introduction

In the last years the number of devices connected to the home networks has drastically increased, with the advent of products like smart TVs, smart speakers, smart meters, smart appliances and other smart home IoT boxes. The proliferation of such devices, in an ecosystem which is often poorly managed but holds very sensitive personal information, raises considerable concerns from the security point-of-view. Recent incidents have shown that these devices cannot be trusted, since their manufacturers often abuse basic user privacy rights to begin with. Moreover, even when manufacturers do behave correctly, the devices are designed with such poor security that they easily become the target of malicious attacks from third-parties, providing a valuable entry point to the wealth of personal information available in the home network, while becoming part of large botnets used for high-profile internet attacks. Those risks are amplified by the fact that the home network is often poorly managed by nature (Cruz et al, 2015), without internal segmentation or restrictions on device-to-device communications, and even without proper security monitoring mechanisms at internal level. Moreover, ownership and management of these devices is often fragmented between different parties (e.g. home owner, other home users, energy utilities, local service providers such as telecommunications operators, and cloud service providers such as Google).

This situation provides the rationale to propose a platform for home LAN security monitoring, providing the means to control which devices may send which information to whom. This platform would monitor in and out traffic, not just to search for malicious activities, like a classic Intrusion Detection System (IDS), but also to control the information exchanged between devices and their manufacturers. The idea is not allowing our personal data concerning our activities, to be sent out. This is a challenging task because first we need to know what that information is, and we cannot count with the manufacturers to help us. So, the first step is monitoring the network to see the kind of traffic that flows and decide what to allow and what to block. After that, we need to produce a set of rules to control the flow of in and out data in an automated way. Also, we need to find a suitable

hardware platform to run all the needed software. The device to perform this task cannot be to big nor power consuming because is to be used in home networks. On the other hand, it must be powerful enough to not introduce noticeable lags in data flow, while keeping a fairly filtering capability of data packets.

This project intends to create such a device for home networks, using a suitable Single Board Computer (SBC) platform for its Proof-of-Concept implementation. As such, this device will be setup between the home router and the rest of the network, so it will be capable of analysing all the in and out traffic. For this purpose, we have chosen the Raspberry Pi B 3+, thanks to its widespread availability, good performance and compact form factor. The OS will be Linux based, and the software to monitor the network will include both a custom solution for anomaly detection, working together with an available IDS platform.

The organization of this paper is as follows: Section II provides an overview of related work addressing the specific security needs of home LANs, also including an overview of relevant approaches and a small survey about the usage of the Raspberry Pi SBC for security purposes. Section III describes the specific requirements for the development of the proposed solution, with Section IV providing insights about the Proof-of-Concept (PoC) prototype implementation. Section V discusses specific implementation details for the network traffic capture and processing modules and, finally, Section VI concludes this paper.

2. Literature Review

Within the home LAN domain, the introduction of automation by means of the IoT paradigm (Amri et al, 2018) can bring several benefits, but also has implications regarding security issues and the privacy protection of objects and users (Panagiotis et al, 2019). Such developments provide a solid argument for providing IDS mechanisms even for small networks (Microsoft, 2014), especially because the latter ones are usually the most susceptible to be abused by all sorts of threats.

In this perspective, various research has been made around packet inspection and traffic analysis using packet sniffers (Asrodia and Patel, 2012) (Qadeer, 2010) these studies are usually intended to be later on used for intrusion detection. In (Rosa et al, 2015) (Cruz et al, 2015) is introduced a new framework to home security trying to fill the gap that currently exists between ISP security approach and home network security. By now these are two completely different “worlds” that do not communicate with each other. This “no one’s land” facilitates the attacks on home networks, because the average user is completely unaware of the dangers. To overcome this, the proposed framework provides a distributed IDS system with residential gateways and ISP IDS systems so they can more easily find malicious attacks.

(Ahmed et al, 2018) proposed a methodology for detecting anomalies that may develop into an attack, which may be useful for our security appliance. In (Ye et al, 2018), a new approach is proposed using emergent technology, in this case software defined networks, to address the problem of monitoring encrypted network traffic. The proposed method uses machine learning algorithms. In (Sperling et al, 2017), is proposed another approach to IDS systems using DNS. The concept is analyzing DNS requests in order to find suspicious sites request. This is also an alternative approach that may help in protect a home network.

In (Aspernäs et al, 2015) the ability for the Raspberry Pi to perform as an IDS is tested. The test comprised two models: Raspberry Pi model B+, and the latest model Raspberry Pi 2 Model B. The IDS software used was *Snort* (Snort, 2018). The result is that the latest model, Raspberry Pi 2 Model B performs better, as would be expected, but also it is the only one that is really able to work as an IDS. In (Mantere et al, 2013) an analysis is made on the specificities of network traffic on ICS networks, and how that affects an IDS performance. (Poonia et al, 2017) also evaluated the need for network IDS in home networks, as well as the capability of the Raspberry Pi to perform these duties. The tests were made with a Raspberry Pi Model 3, a Raspberry Pi Model B+ and a Raspberry Pi Model B, all with the same configuration. The test results showed that using a Raspberry Pi Model 3 with Snort IDS provides a good compromise for a Home network IDS.

In (Kyaw et al, 2015) a comparison is made between *Snort IDS* and *Bro IDS*, as an IDS to be used in a Raspberry Pi 2 for home or small office use. Also, the capability of the Raspberry Pi to accomplish the task was tested. The result is that Snort IDS performs better, but the RaspberryPi2 may crash if there are a lot of traffic on the network. In (Sforzin et al, 2016), another study using Raspberry Pi and *Snort*, is presented reinforcing the opinion that this is a good option for a small sized network security implementation. This paper also proposes a network

architecture that can take advantage of several IDS devices on the same network, which is an interesting area of research. In (Zitta et al, 2018), another IDS is presented, *Suricata*, and its performance is tested in a Raspberry Pi 3. It is another alternative to the already aforementioned IDS's *Snort* and *Bro*. This IDS also performs adequately on the Raspberry Pi platform, although it needs custom rules to be added.

It is important to note that most of the investigation done so far, is about IDS systems, while this paper is about creating a device to control what is going out of a home network, generated by the appliances in the home network. As so, there are no software readily available.

3. Platform requirements

To monitor and block outgoing traffic it is needed what could be called a "reverse IDS"— a device whose purpose is to analyze the traffic generated by the home LAN devices. Considering the fact that even normal IDS roles are not supported by most domestic routers, it comes as no surprise that support for this kind of solution is absent from most home LAN equipment. On the other hand, home routers are "black boxes" so it is not possible to easily add new features. This justifies the introduction of a dedicated security appliance, whose main requirements are:

- It must be inexpensive;
- It must be power efficient;
- It must have a compact form factor;
- It must be easily configurable;
- It must be able to perform traffic monitoring with minimal disruption;

While these requirements may seem somewhat contradictory at first, they are in fact compatible (even for a PoC), especially considering the envisioned deployment domain. Home LANs have fewer requirements in terms of network capacity, which fit within the computing capabilities of several Single-Board Computers.

The last step of the setup is to deploy the device within the home LAN (see Figure 1). It has to be placed like a network-based IDS, between the home gateway and the rest of the network. To do that it will need to have two network interfaces. Also, the wireless traffic needs to be analyzed so the device must have a wireless network interface and be to be configured as wireless AP.

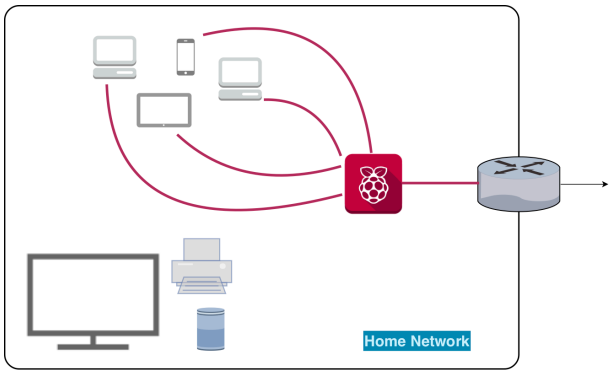

Figure. 1. Deployment of the proposed solution within the home LAN

Regarding software, the OS will be a Linux distribution, and for packet analysis what we need is something similar to an IDS only with a slightly different set of rules. In fact, what the software must do is to look at the packets in and out of the network towards the internet. This is something that is currently done by existing IDS with the only difference been the fact that the IDS look for incoming traffic rather than outgoing. What we will try is to apply the IDS to outgoing packets, filtering the content that the home appliances will send somewhere.

To get to the right set of filtering rules, the first step is to analyze the traffic and try to understand what appliances send out our personal information and capture some packets. Next, is time to analyze the captured packets and try to understand their origin. This is a fundamental step, this analysis will provide the basis for writing the software filtering rules, as well as defining a reference traffic model for a normal state. For the latter

aspect, the use of a conventional NIDS will be complemented with a custom module, which is capable of extracting traffic flow information from the network captures, in order to establish a security baseline.

4. Proof-of-concept prototype

Despite the merits of signature-based approaches (whose benefits are leveraged by the inclusion of a Snort IDS), our purpose is to conceive a solution capable of (semi)autonomously establishing a set of baseline descriptors for the home LAN environment, which will later serve as a reference for security detection procedures. This will be achieved by creating profiles based on the traffic captured from the network. Such profiles correspond to a set of characteristics determined for each device, whose purpose is to identify the communications allowed for that device: with which IP address can the device communicate with, through which ports it can do it, etc.

Profile characterization is based on patterns found when looking at specific features of the captured data such as timestamps and bandwidth. This concept is the core of our solution and what distinguishes it from other security implementations for IoT devices.

4.1 Support Platform (hardware and system software)

To implement the device, our choice fell on a Raspberry Pi 3 B+, the most recent version of the Raspberry family. The most recent model has the following characteristics (Raspberry PI, 2018):

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz, with 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)

This new model of the Raspberry Pi offers adequate computing power, also being a small, inexpensive device, with the bonus of being able to look neat on a household room, thanks to the great variety of cases available for purchase. As the Raspberry Pi came with only one ethernet adapter so a second one had to be added. The solution was to use an USB-Ethernet adapter using one of the available USB ports. For the rest of the setup, the Raspberry Pi came with what was needed, specially the wireless network card.

Regarding the system software, the chosen OS was the ArchLinux Linux distribution (Archlinux, 2018). This distribution was chosen because it is lightweight, having only the core OS structure, and allowing the user to customize the system to their own needs while avoiding wasting resources with non-relevant services. On top of that it will be installed the Snort IDS [15]. Snort works by monitoring and dissecting network traffic data, matching it against a set of rules, trying to find malicious patterns. These rules can be customized, so it may be possible to improve the existing ruleset with custom rules. Although there are other alternatives, like Bro or Suricata, Snort was chosen because is stable, highly configurable, and performs well on Raspberry Pi. The the PoC will also include a custom solution for flow capture and analysis, which will be complementary to the operation of the Snort IDS, described in Section V.

The testing network consists of a home gateway with wireless AP function disabled, the security appliance PoC device configured as a wireless AP , a home switch to allow all the wired links to pass through the appliance (for traffic analysis purposes), a home tv box, two televisions of different brands, and a BluRay reader (see Figure 1). It must be noted that, despite the fact that a home network also includes computers, they are not shown because they are not relevant for this work.

4.2 PoC Network Setup

In our homes, we have two types of devices: the personal ones and the other non-standard devices that connect wirelessly to the network and have the ability to transmit data, also known as IoT devices or smart objects. Any device with an Internet connection can be compromised, which enables third-party vendors being capable of obtaining sensitive data when we connect to their devices or use their applications. Ideally, we want to monitor every device in our network, but the architecture of our initial solution is only intended to capture the traffic of personal devices like smartphones, tablets, and computers.

In order to develop an affordable solution, we used a Raspberry Pi 3 and due to its in-built wireless features, we configure it to act as an Access Point (Figure 1). It was configured as both a wired and wireless bridge – for the latter, the configuration was established as such (Poonia et al, 2017):

Install packages (last version and required software):

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install hostapd
sudo systemctl stop hostapd
```

Edit the file **/etc/dhcpcd.conf**, to configure the **wlan0** interface:

```
interface wlan0
static ip_address=192.168.x.w/24
nohook wpa_supplicant
```

Restart the dhcpcd daemon:

```
sudo service dhcpcd restart
```

Setup hostapd (edit **/etc/hostapd/hostapd.conf**):

```
interface=wlan0
driver=nl80211
ssid=NameOfNetwork
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=PasswordOfTheNetwork
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

After having a device that creates a wireless local area network (WLAN) we had to pass all traffic between the WLAN and the Ethernet interface. We set up a bridge thus making possible for the access point to provide wireless connections: anyone connected to it can access the Internet. It follows its configuration:

Install packages:

```
sudo apt-get install hostapd bridge-utils
sudo systemctl stop hostapd
```

Stop IP allocation for the **eth0** and **wlan0** interfaces (add to **/etc/dhcpcd.conf**):

```
denyinterfaces wlan0
denyinterfaces eth0
```

Add a new bridge and connect the network ports

```
sudo brctl addbr br0
sudo brctl addif br0 eth0
```

Add the configuration for the network bridge (**br0** – add to **/etc/dhcpcd.conf**):

```
Auto br0
iface br0 inet manual
bridge_ports eth0 wlan0
```

Every packet sent and received by any device connected to the Raspberry Pi can now be parsed as a result of the bridge deployed between the wireless interface and the Ethernet connection, as shown in Figure 1.

In the future, this configuration will be improved to cover a bigger number of devices, by configuring a bridge between the internal and an external ethernet adaptor to enable bridging on the cabled network segment. Also, we intend to automate the reconfiguration of the domestic router DHCP server, in order to disable it and offload this task to the security appliance, which will be able to pass its own IP address as the default gateway, forcing the network traffic for all dynamically configured devices to be steered through it.

5. Network traffic capture and processing

Once the Access Point capabilities of the PoC device became operational, we were in conditions to further proceed with its development. Apart from the Snort component (which is a single process component), we have two further concerns: the capture of packets and the data analysis. It is expected that these two main tasks will have different execution times, with the analysis taking longer than the traffic capture. For this reason, we established that each task should be treated by a single process, taking advantage of the fact the Raspberry Pi 3 has a 1.2 GHz 64-bit quad-core processor, to use CPU affinity. CPU affinity enables binding a process or multiple processes to a specific CPU core in a way that the process(es) will run from that specific core only. By dedicating one CPU core to a particular process it is possible to minimize competition with other tasks, ensuring maximum execution speed for that process, with obvious performance benefits.

Setting the CPU affinity is easily done with some lines of code in C:

```
#define _GNU_SOURCE
#include <sched.h>
if (core_id < 0  core_id >= num_cores)
  return EINVAL;
cpu_set_t cpuset;
CPU_ZERO(&cpuset;
CPU_SET(core_id, &cpuset);

int pid = getpid();
sched_setaffinity(pid, sizeof (cpu_set_t), &cpuset);
```

Where `coreid` is the corresponding integer to the core which we want to assign our process (0<**`core_id`**<number of cores-1). Bear in mind that Linux probably runs by default some tasks in core 0.

5.1 Managing parallelism

To benefit from core affinity there is the need to separate tasks among separate processes. As such, the workload for capturing and processing network traces was split among two different processes (one for handling network traffic capture and another one for trace analysis), in order to better decouple tasks and manage buffering. However, by pursuing this strategy there is the problem of making the two processes communicate in a robust and efficient way.

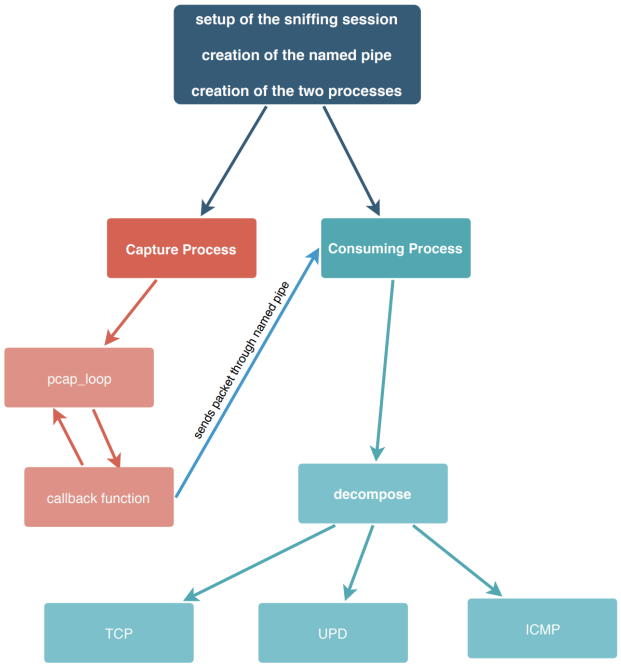

Figure. 2. Trace capture and analysis solution

The first implemented approach to solve this problem consisted on a rotating file pool, using predetermined number of files, with a maximum capture size for each one. When the maximum capture size was reached, the producer process (which captures network traffic) would move to the next file. This approach requires a flow

control mechanism to avoid concurrency problems. As such, a simpler solution using named pipes was adopted (see Figure 2). Also known as FIFO, a named pipe is one of the methods for inter-process communication. A simple `mkfifo` call creates a special FIFO file and any process can open it for reading or writing.

5.2 Network traffic capture

Libpcap (Carstens, 2002) is a library used to capture packets directly from the network adapter, providing the packet capture and filtering capabilities required for the capture module. Figure 2 shows how the solution is structured, i.e how each functionality is organized in the code. We start by doing the required initialization procedures:

Sniffing session

`pcap_open_live()` is used to obtain a packet capture handle to look at packets on the network. One of its parameters is the network device to be open. We are sniffing the bridge that was previously created on the Raspberry Pi. **`pcap_lookupnet()`** looks for the network mask, **`pcap_compile()`** compiles a string into a filter expression and finally **`pcap_setfilter()`** applies the filter. The filter expression restricts the type of packets we want to capture. Currently, our program is capturing all the IP traffic but more strict rules could be applied, for example, if we want we can sniff a specific port or protocol (e.g. the expression "port 53 UDP" would filter all DNS traffic).

Named Pipe

Create the named pipe:

```
#define myfifo 'path/to/fifo/name'
//mkfifo(<pathname>, <permission>)
mkfifo(myfifo, 0666);
```

Open for write/read:

```
int fd[2];
fd[0] = open(myfifo, O_WRONLY);
fd[1] = open(myfifo, O_RDONLY);
```

Processes

After executing **`fork()`**, each process calls its corresponding function where the CPU affinity is set and the FIFO is open either for writing or for reading depending on the process. Then the procedure is different for each function. We will continue the describe the capture process in the present section while the consuming process will be discussed in the next subsection

The capture initializes when pcap enters its primary execution loop that processes packets from a live capture. The responsible function to do so is **`pcap_loop()`**. One of its arguments is the name of a callback function that is called every time a packet that meets our filter requirements is sniffed. The callback function can be implemented to do whatever we need but has a prototype defined in order to **`pcap_loop`** be able to deal with it. These two functions are defined as follows:

```
int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)

void callback(u_char *args, const struct pcap_pkthdr *header, const u_char *packet);
```

The first argument of **`pcap_loop`** is the session handle followed by an integer to define how many packets should be sniffed before returning (we are interested in capture all of them so we set the maximum number as -1 meaning that we want to continually capture until an error occurs), the third argument is the name of the callback function and the last one is reserved for the arguments of the callback function (**`NULL`** in our case). The first argument of the callback function corresponds to the last argument of **`pcap_loop`** the second is a header provided by pcap that contains information about when the packet was sniffed, how large it is, etc. The actual packet is received as a **`u_char`** pointer (last argument) that points to the first byte of the block of data containing the entire packet. The packet is in fact a collection of structures (example in Table 1).

The only purpose of our callback function is to send the sniffed packet (and the header given by pcap) through the named pipe (Figure 2). Whenever a packet is received by the capture process, it is passed it to the consuming pro- cess so that it can be analysed. However, this is done with an auxiliary structure (mentioned below) where

we save the pcap header and the packet as a byte array because we can't send the **`u_char`** pointer through the named pipe (keep in mind that we have two different processes communicating).

```
struct info {
        struct pcap_pkthdr header; u_char packet[SIZE];
};
```

5.3 Data Analysis

As aforementioned, the consuming process sets the CPU affinity. After do- ing so, it enters an infinite loop that is always reading from the FIFO and every time it receives a struct "info" it sends it to a decompose function. We are not concerned with the packet payload i.e, we do not analyze the transmitted data within the actual message. Each packet has an IP header with useful information: source and destination IP addresses, time to live, checksum the protocol. The IP and Ethernet headers are defined for every IP packet but as they are the only thing packets have in common, when the decompose function receives one it does a filtering based on the protocol: TCP, UDP, or ICMP. Table 1 showcases the layout of a TCP packet in memory.

Table 1: TCP packet in memory

Variable	Location (in bytes)
sniff_ethernet	X
sniff_ip	X + SIZE ETHERNET
sniff_tcp	X + SIZE ETHERNET + IP header length
payload	X + SIZE ETHERNET + IP header length + TCP header length

Having the pointer's address, set in the previous table as the value X, we can easily access the other headers and therefore all of the information available in the packet. For example, the length of the Ethernet header (SIZE ETHERNET) is 14 and once we have the packet **`(u char *packet)`** we can access all the in- formation:

Get the IP header:

```
struct sniff_ip *ip;
ip = (struct sniff_ip*) (packet + SIZE_ETHERNET);
```

Determine the protocol:

```
#define IP_HL(ip) (((ip)->ip_vhl) & 0x0f)
int size_ip;
size_ip = IP_HL(ip)*4;
u_char protocol = ip->ip_p;
```

Get the corresponding header:

```
if (protocol == IPPROTO_TCP) {
        struct sniff_tcp *tcp;
        tcp = (struct sniff_tcp*) (packet + SIZE_ETHERNET + size_ip);
}
```

Default values like Ethernet header's length and all of the necessary struc- tures are defined in the code. This paper only covers the identification of TCP flux. This topic is presented in the following subsection.

TCP flow parsing

TCP is a transport layer protocol used by various services such as FTP, SMTP, HTTP and many others. This byte-stream-oriented protocol sets up a connection to the receiver using the three-way handshake method and then sends the data in segments. Sequence numbers are attributed to the packets in order to keep track of the bytes sent in each direction.

The TCP header is defined as follows:

```
struct sniff_tcp {
        u_short th_sport;
        u_short th_dport;
        tcp_seq th_seq;
        tcp_seq th_ack;
        u_char th_offx2;
```

```
        #define TH_OFF(th)
        u_char th_flags;
        #define TH_FIN 0x01
        #define TH_SYN 0x02
        #define TH_RST 0x04
        #define TH_PUSH 0x08
        #define TH_ACK 0x10
        #define TH_URG 0x20
        #define TH_ECE 0x40
        #define TH_CWR 0x80
        #define TH_FLAGS (TH_FIN|TH_SYN |TH_RST|TH_ACK|TH_URG|TH_ECE |TH_CWR)
        u_short th_win;
        u_short th_sum;
        u_short th_urp;
};
```

We have information about the source port, destination port, sequence number, acknowledgement number, data offset, flags, window, checksum and urgent pointer. These parameters about the TCP packet will most likely be used in a further analysis, but by now, a fairly simple identification of TCP flux can be done by the combination of the Source IP address, the Destination IP address and TCP port numbers (both source and destination).

We maintain a record of the TCP Streams in a linked list. Each node is defined by the four parameters aforementioned (source and destination IP addresses and source and destination ports).

```
typedef struct node_tcp {
        char from[100];
        char to[100];
        int src_port;
        int dst_port;
        struct node_tcp *next;
} tcp_stream;
```

A method for searching the list was implemented such that, if we look for a node with a specific combination of addresses and ports and that node already exists then it is put in the first position. Whenever we received a large number of packets of the same stream in a row, the time to search/access the node is the lowest possible.

6. Conclusion and Future Work

This work explores a new perspective about home network security. Here, we are not yet trying to avoid attacks from the outside, although it stills a major concern, but rather figuring out if and how, our home appliances are giving our personal information to third parties without our permission. To achieve this, a thoroughly network analysis must be performed in order to understand what kind of information is sent from our home appliances to the outside. When we will be able to define the pattern behavior it will be possible to implement a traffic analyzer to enforce that same behavior and avoid having our personal information to be sent out.

For the aforementioned purposes, the establishment of a reliable capture and interpretation of packets is the first step to design our security appliance. This makes it possible to perform a concrete analysis of each packet, more specifically about each TCP communication. The main goal is to find meaningful patterns that allow us to establish a set of rules for each device. The ultimate objective of this effort will be the creation of a dynamic process capable of detecting a new device in the network and active the proper “profile” for it thus enabling a set of communication rules based on the previous analysis.

Further developments of this platform will also include support for virtualized home gateways (Cruz et al, 2013), for which the authors envision an integrated approach, using Virtualized Network Functions (VNFs) to host the functionality within virtualized service instances.

7. Acknowledgements

This work was partially funded by the ”Mobilizador 5G” P2020 Project (project 10/SI/2016 024539) and the ATENA H2020 EU Project (H2020-DS-2015-1 Project 700581).

References

[1] Ahmed, A. A. and Kit, Y. W. (2018) Collecting and Analyzing Digital Proof Material to Detect Cybercrimes, Faculty of Computer Systems & Software Engineering, Universiti Malaysia Pahang.

[2] Amri, Y. and Setiawan, M. A., (2018) Improving Smart Home Concept with the Internet of Things Concept Using RaspberryPi and NodeMCU, Department of Informatics, Universitas Islam Indonesia, IOP Conf. Series: Materials Science and Engineering 325.

[3] Arch Linux for Raspberry Pi, [online], https://archlinuxarm.org/platforms/armv8/broadcom/raspberry-pi-3

[4] Aspernäs, A. and Simonsson, T. (2015) IDS on Raspberry Pi – A Performance Evaluation, Diploma Thesis

[5] Asrodia P. and Patel H. and Network Traffic Analysis Using Packet Sniffer, 2012

[6] Carstens, T. Programming with pcap, 2002.

[7] Cruz, T. and Simões, P. and Monteiro, E. and Bastos, F. and Laranjeira, A., "Cooperative security management for broadband network environments", *Security Comm. Networks*, 8: 3953– 3977. doi: 10.1002/sec.1313, 2015.

[8] Cruz, T. and Simões, P. and Reis, N. and Edmundo Monteiro and Bastos, F. and Laranjeira, A. , "An Architecture for Virtualized Home Gateways ", in IM 2013 (IFIP/IEEE International Symposium on Integrated Network Management), 2013.

[9] Kyaw, A. K. and Chen, Y. and Joseph, J. (2015) Pi-IDS: Evaluation of Open-Source Intrusion Detection Systems on Raspberry Pi 2, IEEE.

[10] Mantere, M. and Sailio, M. and Noponen, S. (2013) Network Traffic Features for Anomaly Detection in Specific Industrial Control System Network, VTT Technical Research Centre of Finland.

[11] Microsoft (2014), Noticing and Responding To Network-Borne Attacks, [online], https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc723457(v=technet.10) .

[12] Panagiotis I. Radoglou Grammatikis and Panagiotis G. Sarigiannidis and Ioannis D. Moscholios, Securing the Internet of Things: Challenges, threats and solutions, Internet of Things, Volume 5, 2019, Pages 41-70, ISSN 2542-6605, https://doi.org/10.1016/j.iot.2018.11.003.

[13] Poonia, P. and Kumar, V. and Nasa, C. (2017) Performance Evaluation of Network based Intrusion Detection Techniques with Raspberry Pi - a Comparative Analysis, International Journal of Engineering Research & Technology (IJERT), ICCCS.

[14] Qadeer M. A., Iqba and A., Zahid and M., Siddiqui and M., Network Traffic Analysis and Intrusion Detection using Packet Sniffer, 2010

[15] Raspberry Pi specifications, [online], https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[16] Rosa, L. and Alves, P. and Tiago Cruz and Simões, P. and Edmundo Monteiro , "A Comparative Study of Correlation Engines for Security Event Management", in In Proc. of 10th Int. Conf. on Cyber Warfare and Security (ICCWS-2015), 2015. ISBN: 978-1-910309-98-8 ISSN: 2048-9897.

[17] Sforzin, A. and Conti, M. and Mármol, F. G. and Bohli, J-M.. (2016) RPiDS: Raspberry Pi IDS A Fruitful Intrusion Detection System for IoT, 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress.

[18] Snort website [online], https://www.snort.org/

[19] Sperling, T. L. von and Filho, F. L. de C. and Júnior, R. T. de S., Martins,and L. M. C., and Rocha, R. L., (2017) Tracking intruders in IoT networks by means of DNS traffic analysis, 2017 Workshop on Communication Networks and Power Systems (WCNPS).

[20] Ye, F. and Qian and Y. (2018) DataNet: Deep Learning based Encrypted Network Traffic Classification in SDN Home Gateway, IEEE Access.

[21] Zitta, T. and Neruda and M., Vojtech, L. and Matejkova, M. and Jehlicka, M. and Hach, L. and Moravec, J. (2018) Penetration Testing of Intrusion Detection and Prevention System in Low-Performance EmbeddedIoT Device, 18th International Conference on Mechatronics, December.