University of Lorraine

Master 2 internship report

David Cruciani
Année 2020-2021

University tutor : Mme. Herrmann
Company tutor : M. Dulaunoy

1 April 2021 — 30 September 2021

Contents

1 Special Thanks

I would like to thank the company SECURITYMADIN.LU which allowed me to carry out my end-of-study internship in order to validate my Master 2 computer science option security of information systems. A special thank to Alexandre Dulaunoy, my internship supervisor, as well as to Jean-Louis Huynen who assisted Mr. Dulaunoy in my follow-up, for the confidence they have given me in the tasks I have had to perform, and in the help, as well as, in the advice that I have been able to receive from them. Finally, a special thank to all the staff who answered my questions and gave me help when I needed it.

I also want to thank Anselme Morisot who did his internship in the same company, we were able to work on call sometimes when we were telecommuting and he went to the office the same day as me on several occasions when we weren't working remotely.

Finally, I would like to thank the University for the past 5 years of training at the UFR MIM in Metz and especially our two supervisors of this last year Mrs. Herrmann and Mr. Lanuel.

2 Introduction

As part of my Master 2 computer science option information systems security, I had to complete an internship of at least 18 weeks to validate my grade during which I met Mr. Dulaunoy through the intermediary of a course he gave to the entire promotion. This course, dealing with Threat Intelligence * and a bit of Forensic *, was most fascinating. Following this, I contacted him to find out if it was possible to do an internship in his company, then he suggested a subject: Profiling user activities and usage based on forensic artefacts. This subject was subsequently modified, we agreed on a new subject and I was able to start my internship on April 1, 2021 to finish it on September 30 of that same year; the internship will have lasted practically 6 months. As said previously, the subject has been modified but it will be presented in more detail in another part, it is nevertheless, like the one stated previously, oriented towards Forensic * and more precisely in the creation of tools allowing to carry out these actions.

The internship therefore took place with Mr. Dulaunoy in the company where he works, SECURITYMADEIN.LU in the CIRCL department, a large part took place remotely due to the Covid-19, I was not able to , to this date, meet all employees, physically or remotely.

In order to continue this report and to present my subject and its development in the best possible way, a presentation of the company and the working environment will be made beforehand. Subsequently, I will present possible points for improvement after these 6 months of internship, then I will end with a conclusion.

3 Company

SECURITYMADEIN.LU is an Economic Interest Group*, contributes to the Luxembourg economy's trustworthiness by providing extensive cybersecurity expertise and solutions to SMEs* and companies of all sizes as well as Municipalities through its 3 departments : *CIRCL* (Computer Incident Response Center Luxembourg), *CASES* (Cyberworld Awareness Security Enhancement Services) et *C*3 (Cybersecurity Competence Center). *CASES* is focused on raising awareness of IT security issues in general while *C*3 is focused on improving cybersecurity skills already in place.

CIRCL (Computer Incident Response Center Luxembourg) is a government-driven initiative designed to provide a systematic response facility to computer security threats and incidents. It's the CERT for the private sector, communes and non-governmental entities in Luxembourg.

provides a reliable and trusted point of contact for any users, companies and organizations based in Luxembourg, for the handling of attacks and incidents. Its team of experts acts like a fire brigade, with the ability to react promptly and efficiently whenever threats are suspected, detected or incidents occur. CIRCL's aim is to gather, review, report and respond to cyber threats in a systematic and prompt manner.

The team is made up of 14 members working on information security and three members who work on the legal and business part of the department.

CIRCL is a CERT accredited by *Trusted Introducer*, an organization grouping together in a directory all the teams accredited by it to offer trustworthy services.

Here are some services offered by CIRCL:

- Reporting of security incidents;
- Incident identification, triage, analysis and response;
- Technical investigation:
  - Incident correlation;
  - Malware analysis and reverse engineering;
  - System and network forensic analysis;
  - Security vulnerability assessment;
  - Information leak analysis and data mining;

CIRCL also offers tools to help investigations mainly of the type:

- URL abuse, allowing to check the reliability of URLs;

- cve-search (Common Vulnerabilities and Exposures (CVE)), the objective is to be able to find vulnerabilities of a given software;
- MISP (Malware Information Sharing Platform), an information sharing platform.

The internship therefore took place in the SMILE company (SECURITYMADEIN.LU), in the CIRCL department. The head of department, Mr. Dulaunoy, has been my tutor during the past six months, I was not assigned to a particular team since the subject I dealt with was reserved for me and should not require more six months to be completed.

4 Subject presentation

The original subject consisting into Profiling user activities and usage based on forensic artefacts having been modified, the use of artefact * remains however still present and represents one of the most important parts: use remarkable artifacts * to determine whether software has been installed or is installed on a machine.

This topic may seem unnecessary at first sight, because in a computer it is easy to list the installed software. However, it is necessary to understand the context of its use or at least the purpose of its development: When a company suffers an attack, it has become normal to call a CERT (Computer Emergency Response Team) or more precisely an incident response team in order to determine the causes and consequences of the attack and if there would still be traces of the attacker in the company network. To understand the attack, a hard drive is often acquired, but if the victim PC is not identified and only the software that allowed the attack is known, then this subject becomes interesting.

A concrete example, told and experienced by Mr. Dulaunoy is the following: A company has been attacked and informs him that the problem comes from a specific sector and that the malicious program was downloaded from a computer using *Chrome* browser. The problem is as follows: in this department, there are 10 workstations, so we have to recover the hard disk of each one, a step that takes a lot of time but is unavoidable, then we have to analyze each machine individually. Thus, a lot of time is lost in identifying the victim machine. Each hard disk is recovered and then analyzed more or less thoroughly. Having help to find out what software is installed can be a great help.

The primary goal of this project is to save investigators' time by providing a means of triage in complicated situations as described above. If I had to quote Alexandre Dulaunoy to summarize the subject of my internship, it would be: "it's for sorting my shit".

To determine if a software is or has been installed on a machine, I used Yara, a tool that resembles in many ways a programming language but rather focused on regular expressions. Yara is used by creating rules and I use remarkable artifacts* that are on the machine that must be retrieved first, to complete them. Once created, these rules will be applicable directly on the hard disk recovered from a machine, the time saving is done at this moment.

To obtain the artifacts*, you have to install the software on a machine, take out the information that will be used to obtain them, uninstall the software and repeat this process for each software. These repeated actions are extremely time consuming if everything has to be done by a human and several hundreds of software have to be installed. So I focused on creating rules in the most automatic way possible.

I carried out this project alone without a team but with Mr. Dulaunoy and Mr. Huynen who supervised, guided and helped me. As a result, I was able to move forward at my own pace without having any particular pressure. On the other hand, I think that with a team the result can be more thorough and better.

Concerning the writing of the code, the whole is done in *Python*, choice made for the ease of writing and for the great diversity of library very quickly accessible. I also used some tools already created and available on the internet which will be presented throughout the development of the subject, but the base of the program and its biggest content was written by me from scratch.

Two repositories have been created on *Github*, one is used to gather the created rules [1] and the other one is used for the code doing the generation [2].

5 Réalisation du projet

This part is dedicated to the realization of the project, the latter has been recalled just above, but a reminder is never too much, the subject is therefore: use remarkable artifacts * to determine whether software has been installed or is installed on a machine.

5.1 Formation

As the whole beginning of the training took place remotely, I didn't have a laptop to work with but I had an SSH* access to a server allowing me to test the different programs produced. The use of this remote server will of course be described later.

Before starting to work on the topic of the internship, I had a small self-training part which was split into two parts [3]: reading the 3 pdf given first, and second, use of the tools presented in the three different courses. The reading took place with a note of the most important points for me in order to have easy access to an information in need, but also because it was a request of the tutor [4]. For the practice, I pushed a little more the use of the tools of the courses by using other tools and in different places, I notably used a poster of *SANS* (SysAdmin, Audit, Network, Security) that I transcribed, but only the parts having a real interest for the project in order to have a better access [5] and to be able to make some additions that seemed to me useful for the project. The tools are always free and often, see every time, retrieved from *Github* and may be more or less recent and more or less known to the general public. The most complicated tools to use are always those that will work on the *registry**, the place where a lot of information of the most useful for an investigator is stored, but the best known is *regripper*. It often happens that the tools available for this use are created to be used under Windows, which was not the case for me during the analyzes.

This introduction was quite formative on the basics of Forensic * and lasted about a week.

5.2 Analyse de Disque

This sub-part is dedicated to some basic principles that I was able to learn during my training but which will be necessary to fully understand the rest, it is also the methodology that I applied at the beginning of my treatment of the subject. It is also found in the main part which is automation, part described just after.

5.2.1 Acquisition

When we want to analyze a hard drive, it is essential to know certain things: a drive is never analyzed on the spot, that is to say, on the machine which would have suffered an attack. Knowing this, it is therefore necessary to acquire the disk in order to be able to analyze it off site, but above all and mainly to be able to keep the disk as intact as possible. The fact of using the victim machine will necessarily write to the disk, even if you do not copy files or save nothing to the disk, but simply because the operating system will clean up or will write to disk for updates. In my case, in order to be able to make a small and simple acquisition, I used a Windows virtual machines (VM), so that a copy and paste will be the acquisition action to allow the main disk to not be touched, knowing that the copy will also be modified as little as possible.

5.2.2 Conversion

Once the acquisition is made, a conversion may be necessary. In real life, the disc is already in the correct format when acquired because most tools will do the job as a whole. In my case, I do not use acquisition tools but rather a conversion tool called $qemu - img$ and which allows to convert a virtual disk into a large exploitable file for analysis, while being careful not to modify anything in the data of the disc, its use then becomes simpler and above all possible. To limit the different processing times, the disk used is not large, it does not exceed 35 GigaBytes.

5.2.3 Préparation

After the conversion comes the part which I will call preparation. Here it is interesting to choose the tools that will be used so that in the next part you can focus only on the analysis, although it is not always possible to predict all the tools to be used. It is also important to be able to access the drive as if a USB stick were inserted into the machine, since the drive is currently a single file with the raw data inside. To do so, you just have to use a Linux VM containing the appropriate tools, which is my case, or software under Windows allowing to do the same thing. Under Linux, there is a command called *mount* which allows to "mount" a disk or, in other words, to make it accessible and that we can browse it like a classic folder, and that from our file obtained by conversion. Once again, care must be taken not to touch anything on the disk and there are several ways to do this: I, personally, used an option of the *mount* command in order to protect the files in the disk against writing, it is the *readonly*, to prevent any modification of information which could distort the result of the analysis.

5.2.4 Analyse

Once the disk is accessible, it is possible to use different tools to recover artifacts * on the disk, but in general it is necessary to extract from the disk the files likely to contain artefacts *, such as for example logs, to then apply the software to it. In my case, during training, the only tools that applied directly to the mounted disk were to work on the *registry** or to retrieve the disk tree. The elements on which I applied myself to analyze are the Windows logs, a part of the *registry** that it is possible to extract (NTUSER.dat), the *jumplist* which are some shortcuts and the shortcuts created automatically by the machine compared to recent locations visited. In case a browser was installed, the browsing history was also collected.

5.3 Automatisation de création de règles Yara

This sub-part is dedicated to automation, it is the most important part of this report.

As said previously, the method described above will be used in automation at different stages of its development, but this will be specified during the development of this part.

To put it into context, so far the work done has been done on a "mounted" disk but the end goal was to work on a raw disk without having to *mount* and try to extract artifacts * using tools after recovering the interesting files. In appendix 1 is a screenshot of what you can get on disk using the *xxd* tool, and you can see that it is not very tidy and difficult to read. By deviating a little from the methodology, I preferred to look for tools capable of helping me. Fortunately tools like *TheSleuthKit* (TSK) have been created for this specific purpose and allow you to see a little more clearly (Appendix 2). It is with these meager weapons that I launched into the subject, starting, after choosing a few tools, with the automation of VMs.

5.3.1 Automatisation des VMs

In order to run the program without human intervention, it was necessary to ensure that the VMs could start and shut down automatically.

For startup I used *VBoxManage* which comes with VirtualBox *. Knowing that once the VM is created it is registered by VirtualBox and therefore thanks to *VBoxManage* I can find the list of VMs on the way or registered for example. I can also start or stop a VM, using *VBoxManage startvm MyVirtualMachine*. Once started, it is necessary to wait until the machine is switched off before proceeding to the next step and being able to use the results produced. As before, we use *VBoxManage* with *list runningvms* in order to know if our machine is still running. Here is a small algorithm to summarize this first step:

```
1 si The machine is not already running alors
2     Start the machine;
3 fin
4 Wait for the machine to switch off;
```

Algorithme 1 : Automatisation VM

Now comes the acquisition and conversion phase. Because of using a virtual machine, the acquisition results in a simple copy and paste of the *vmdk* or *vdi* file or even by no manipulation. The conversion, on the other hand, is present and is done as presented in the methodology, with $qemu - img$.

Working on the remote server, I had to deploy virtual machines in order to reproduce what I had already done locally but also to be able to use a more powerful machine. The problem was to create everything without an interface since the server did not have a graphical interface. Using mainly a site called [6], I learned to set up VMs without an interface. The main tool is again $VBoxManage$, which allows you to create, manage and delete virtual machines without too much difficulty but with some subtleties especially for VMs that are launched without an interface because in order to be able to access them anyway, you have to add additional configurations concerning the VRDE (VirtualBox Remote Desktop Extension). Knowing that the VM will be launched as shown above, $VBoxManage$ *startvm* $MyVirtualMachine$, we add at the end of the command $--$ *type headless* so that the VM is launched without an interface, otherwise an error message is sent anyway, indicating that it is not possible to launch the machine with an interface. Once the machine is launched without an interface, it can still be accessed through the port specified during the VRDE configuration. However, it is the server that launches the command, so it is the server that could access it through the VRDE protocol, but by using PuTTY properly it is possible to create an SSH* tunnel that will allow my local machine to access the machine launched on the server (Appendix 3). It is necessary to know that the VRDE protocol only works locally by specifying an access port, that is why it is necessary to set up an SSH* tunnel.

A small script written in *bash* allowing to perform these actions has been created by me, as I have the need to create and delete easily and quickly a VM. It will resume all the creation steps by copying a Windows virtual disk with some modifications and already with the operating system installed (Windows). So all you have to do is specify the name of the machine you want as a parameter and then add *del* as a second parameter if you want to delete the VM with this name.

5.3.2 Automatisation de l'installation

After the virtual machine part, comes the software management part for the installation and uninstallation on a virtual machine. The solution brought for this part evolved throughout the project since while advancing, new problems were presented, implying to find new possibilities. To understand what is done here, we try to give a task to the virtual machine, created just before, so that it installs or uninstalls a given software, with the idea to have as less human interaction as possible.

The first choice was *chocolatey*. It is a tool that allows you to install software as you would do on a Linux machine with $apt - get$, that is, with a list of available packages, with little human interaction and on the command line. It's possible to find on the web site a list of all available packages [7], and with a simple command, the installation is done: *choco install* $-y$ *putty.install*, like uninstallation: *choco uninstall* $-y$ *putty.install*. That's all for software management. Concerning the interaction part, I started with $flask$ a python library to install to make a basic web server. The software to be installed by the VM was written on the *installer* page and at the next connection, a redirection was made to the *uninstaller* page allowing the machine to understand in which phase it is. Only, as I progressed, I was asked to add the possibility of choosing the way of installing software and this because using *chocolatey* has disadvantages that are still present today: Not all software is available on this platform and in the Yara rules, which we will see later, there were some additions due to the package to install. This problem led to the removal of the $flask$ server and introduced the installation using *msiexec*, a tool integrated in Windows allowing the management of *msi* software, which avoids human interaction but only works on *msi* files. I also added the ability to install an *exe* file but it requires almost total human interaction to install and uninstall. As said before, I had to delete $flask$ because to use *msiexec*, you need the *msi* file and to avoid the machine in charge of the installation to have to make a request on the web, I set up a shared folder between the physical machine and the virtual one allowing to place the *msi* and *exe* files in a folder.

Concerning the resolution of tasks on the VM, I created a client, which has changed a lot throughout the project, whose purpose is to do the installation and to perform other tasks, which I will detail in the set, necessary to obtain better Yara rules. I'm talking about client because at the beginning when using $flask$, there was a real $server - client$ system. Currently, this system is less present but the program on the VM will always receive "orders" to do. This client will read a text file, that the program having launched the VM will have written, in which will be specified the name of the program to install, the name of the main *exe* file once the installation is finished and finally the way the installation must be done. Here is an example: $putty.msi : putty, installer : msiexec$. The client will know if it has to install or uninstall thanks to the name of the text file that will describe what it has to do. Then, for an installation, it will look for the location of the program with the name that we will have specified to be able to copy the *exe* file out of the machine, to create a Yara rule, and to be able to run the program to get more trace of the software. Launching the software is important because it is assumed here that the software would also be launched on a machine in a real situation, so it is important to have the maximum of traces.

For the uninstallation, I found myself facing a few more problems that cost me quite a bit of research time. The problem with uninstallation is that it is not enough on its own to be effective in the Yara rules created. When we try to remove a program, traces remain and this is normal since in our process we want to recover them, however, with a simple uninstallation, the files are not really deleted but announced as such by the system. The place where these files were then becomes accessible to future files but as long as there is no writing over them, they remain present. Therefore, we have to do a little more to not have the same rules as for the installation. For that, I tried several things, starting with *Sync* [8]. This tool will empty the cache that the system keeps, that is, sometimes the system does not make changes on the disk directly, it keeps them in cache to make them later. Thanks to this, less software residues are found in the generated rules. But two problems occurred, the first one is that there is more than enough left and therefore *Sync* is not necessarily enough to limit the residues. The second one is that when I tried to solve the first problem, I didn't get any logical result because I had the same number of rules as for the installation. The purpose of this modification, which in the end did not bring any results, was simply to write to the disk, since the space became available. This process should have crushed the remaining residues and left only the most persistent traces, which are then of great interest and valuable artifacts*. I tried to show the problem by setting up logs of the various important commands, and if each time everything went well, in the end the rules were not good.

Mr Dulaunoy then advised me another tool, *SDelete* [9]. This one will not empty the cache but it will do much better, it will purge the table that manages the available spaces, because in this table, a deleted file will have a star after its name and its location is also indicated, and the tool will erase what is in the indicated locations. After cleaning, the results for the Yara rulers become very interesting because there is very little residue left and only the important traces are present.

The last point I want to make about the program that runs on the virtual machine, the client, is the following: I have been asked to add a feature to the program, which is to identify the installed files to be able to do a *md*5 and *sha*1 hash and then use *Hashlookup* [10]. The idea here is to be able to identify known files and to be able to determine if a file is corrupted thanks to its hash. To identify if the hash is known, I use, as mentioned just before, *Hashlookup* which is an API created by CIRCL allowing to check if a hash exists in the available database (Appendix 4). Concerning the part of collection of the installed files, I used *ASA* (AttackSurfaceAnalyzer) [11]. This tool will make a point of comparison before the installation and another one after, making the difference between the two points, the created, modified and deleted files are identified and specified in a file in *Json* format gathering all these data. The created report is extracted from the VM and only the information we are interested in is retrieved, namely, the part concerning the created files and especially their location. The sorting of the information is no longer done by the client because the report is taken out of the VM, so it is handled by the main program. Once the access paths are recovered, the disk is mounted, as seen in the methodology, in order to access the created file and to perform a hash on each of them. The hash is performed with *md*5*sum* for the hash *md*5 and with *sha*1*sum* for the hash *sha*1.

To summarize the actions of the client on the virtual machine, here is a small algorithm:

```
Read the task to do;
si Uninstall alors
    appManager(); //uninstallation
    SDelete();
sinon
    AsaCollect(); //point of comparaison
    appManager(); //installation
    Search for exe path;
    copy of exe;
    run of exe;
    AsaCollect(); //point of comparaison
    AsaExport(); //creation of the report
fin
Shutdown
```

Algorithme 2 : Client

5.3.3 Yara

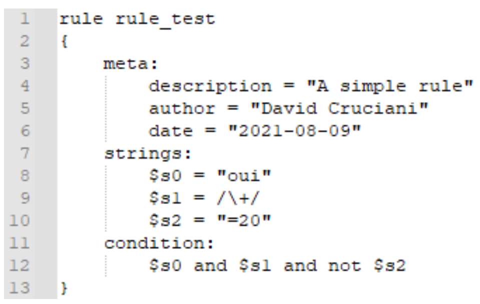
In order to understand Yara and how it works, I will give a part to its explanation.

Created by VirusTotal, Yara is a tool used in the detection, research and identification of malware. Its operation is based on the description of a malware in the form of text, regular expressions, hexadecimals etc...

For this project, no malware is sought but known and legal software, however a description of the software is done by the collected artifacts*.

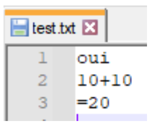
In a Yara file, in addition to the name of the rule, there are three parts: the first one contains information that is not taken care of during an execution, but that will be used to describe what is done, who does it, and when. The second part contains the rules that will be used to describe the software, in our case. They can be written in different ways, as said before, as text, as regular expressions or as hexadecimal. Finally, the third part will define how the rules defined above will be used, i.e. we can specify a threshold, for example, to say that if the threshold is reached then the rule works. Another possibility would be to say that a rule must not be present for the file to be compliant. Another way to do this is to specify the use of a variable entered by the user during its use, allowing for example to make a changeable threshold in order to perform different tests.

In this example, 2 types of writing are defined, the normal text with quotes ($s0 and $s3) and a regular expression with slashes ($s1). We find the three parts described above with *meta*, *strings*, and *condition*. The



```
rule rule_test
{
    meta:
        description = "A simple rule"
        author = "David Cruciani"
        date = "2021-08-09"
    strings:
        $s0 = "oui"
        $s1 = /\+/
        $s2 = "=20"
    condition:
        $s0 and $s1 and not $s2
}
```

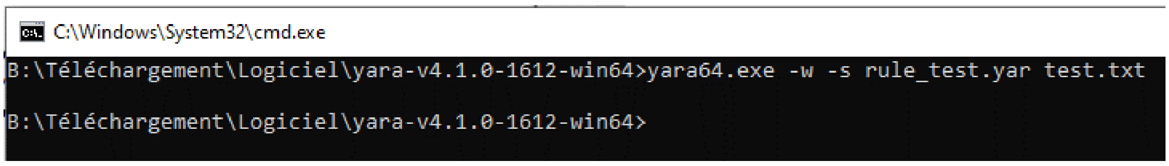
name of the Yara rule is on the first line and this is what will be displayed if the rule works on a given file.



```
test.txt
1  oui
2  10+10
3  =20
```

Here we have a text file on which we will apply the rule that has just been created, in order to see the result that will give the condition that is applied in this rule.

The result is null because the condition does not match with the file.

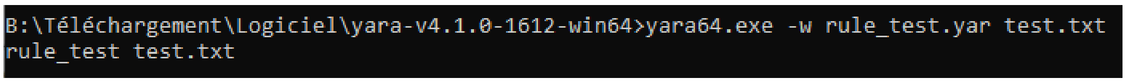


```
C:\Windows\System32\cmd.exe
B:\Téléchargement\Logiciel\yara-v4.1.0-1612-win64>yara64.exe -w -s rule_test.yar test.txt

B:\Téléchargement\Logiciel\yara-v4.1.0-1612-win64>
```

Taking the same test file and the same rule and changing the condition, the result will be different, the condition will be: *all of them*.

The rule has worked this time on the file *test.txt* and the name of the rule that is defined is displayed in front of the name of the file that has been recognized.



```
B:\Téléchargement\Logiciel\yara-v4.1.0-1612-win64>yara64.exe -w rule_test.yar test.txt
rule_test test.txt
```

5.3.4 Création de Règles Yara

Now that the basics about Yara are laid out, we need to understand how the rules are obtained.

Yara rules are the final result of the created program, it is by applying them on a hard disk that it is possible to tell if a software is or has been installed on a machine.

Before you can create them, you have to collect artifacts*, evidence that the software is installed in the most efficient way possible. Earlier in the report there was some explanation about the client and it was said that the *exe* file of the installed software is extracted from the virtual machine in order to be able to make a Yara rule on it. This is the first collection that is done in this process. So now we have to present the other two collections made in order to complete the understanding of the created program.

The two collections are done in a row and each one uses a tool present on Linux and which is more than used in the field of Forensic: *strings* and *fls* a command from *The Sleuth Kit* (TSK). *strings* will return all readable strings in a file passed as a parameter (Appendix 5) and *fls* will return the machine tree (Appendix 2), the list of all files present in the machine. Both will be applied to the result disk of the conversion, described above.

The first command run on the disk is fls which gives the whole tree structure, then the result file is used to create rules, the command is applied for installation and uninstallation. This file gives good indications about a software and just with its name it is possible to find many files installed by the software. The second command is *strings*, but not only that, another tool is used in addition which is *grep*. This will allow you to give a term and limit the result to the term passed in parameter. On a hard disk, so many things are stored that having all the possible strings requires too much processing time to create the rules afterwards. Once the two commands have been completed, the creation can begin.

In the written programme, these two steps do not follow each other, but in order to understand the creation, the rest is put aside.

To resume, two files are retrieved with two commands, one of the files contains the tree structure of the machine, and the other contains the readable strings. From these results, two types of rules will be created, tree rules and disk rules. As these two types of rules are not obtained in the same way, it is not possible to make a single common rule. They cannot be applied in the same way during investigations, because since the tree structure is obtained from fls, the investigator will have to redo the same command and then apply the rule specially created for it to the result. This will allow more possibilities when using it and especially more mobility, as its execution is extremely short compared to the disk rules, both to create and to use them. So if the program is detected in the tree structure it is not necessarily useful to apply the disk rules.

For creation, the resulting files are run through processing to avoid false positives and duplicates.

The files are opened and read line by line, for the result of *strings*, each line will be tested against the desired program name, quite similar to *grep* but the "options" are different. Indeed, when the *grep* command is given, the $-i$ option is applied, indicating that the search should not be case sensitive, the letters can be upper or lower case, while during processing, the test is done on the name without touching it, putting it completely in lower case, then completely in upper case. Let's take the example of $PuTTY$, with the *grep* $-i$ command, there will be in the result lines corresponding to *inPuttype* or $InPuTyPe$... Therefore, it is necessary to apply the treatment on it with $PuTTY$, *putty* and $PUTTY$, which are the main writings of the software. Another problem is the size of the lines obtained, some results are found in a program where everything is compact and fits on one line and is unreadable. To overcome this problem, each line is tested in its size so as not to exceed a threshold and to have readable and understandable results. Then, each line is checked for having never been seen in what has already been read thanks to a table that will add each conforming entry to create a rule just after, an important step to avoid duplicates. Finally, when analysing a vanilla machine, one that has just been created, some software already has traces in the machine because of a list that is already there. The solution to the problem is to have the result of *strings* from a vanilla machine in order to do a comparison on each line, an important but time consuming step. So I added a step at the beginning of the global program consisting in doing the *strings* command with $grep\ -i\ -E$ on the result of *strings* on the machine. The $-E$ option of *grep*, allows to make extended regular expressions but what we are interested in is to specify the list of software that will be analysed in order to greatly reduce the size of the original *strings* file and to greatly improve the rule creation time.

Regarding the result of fls, the same treatment is applied, the name follows the same logic with base name, completely lower case and completely upper case. The size of the entry is not checked as it is a file name and therefore it is very unlikely to have very long file or folder names. The duplicate check is done in the same way, with an array, and finally a comparison is made with the result of fls on the vanilla machine, it is possible to make a smaller fls file as above, to limit the comparison entries, however the file size is not very large and hardly slows down the processing.

Once the processing is complete, an array is created containing all the compliant and unduplicated entries. This array is passed to another function that will create the rule itself, each box of the array will be a rule to describe the software, they will be in the second part of the Yara rule. Each box will be changed into regular expressions to make sure that each character will be taken into account in the application of the rule, to avoid a character being misinterpreted. The condition of the Yara rule is going to be rather basic: a certain match threshold must be reached to validate. I had, at first, put the threshold at an error rate of 25%, i.e. the number of rules was divided by 1.25 to obtain the threshold, the idea is to limit the impact of badly formed rules. Currently this value is an external variable that must be specified by the person who is going to use it, which can allow to find the smallest traces of the software, by setting a very low threshold, or to ensure its presence by setting a high threshold.

The rule is then saved on the disk and that's it.

Here again is an algorithm to summarise the steps taken:

```
1  reading = empty;
2  flsResultat = empty;
3  fichierAAtraiter = reading the file to be processed;
4  si a strings file exists alors
5  |  // Either strings vanilla machine or strings software;
6  |  reading = read the file;
7  fin
8  si fls file of the vanilla machine exists alors
9  |  flsResultat = read the file;
10 fin
11 pour i from 0 to fichierAAtraiter faire
12 |  si fichierAAtraiter is not a result of fls alors
13 |  |  si read is not empty alors
14 |  |  |  si fichierAAtraiter[i] is valid and with reading alors
15 |  |  |  |  Add fichierAAtraiter[i];
16 |  |  |  fin
17 |  |  sinon
18 |  |  |  si fichierAAtraiter[i] is valid alors
19 |  |  |  |  Add fichierAAtraiter[i];
20 |  |  |  fin
21 |  |  fin
22 |  sinon
23 |  |  si flsResultat is not empty alors
24 |  |  |  si fichierAAtraiter[i] is valid and with flsResultat alors
25 |  |  |  |  Add fichierAAtraiter[i];
26 |  |  |  fin
27 |  |  sinon
28 |  |  |  si ffichierAAtraiter[i] is valid alors
29 |  |  |  |  Add fichierAAtraiter[i];
30 |  |  |  fin
31 |  |  fin
32 |  fin
33 fin
34 CreationRule();
35 SaveRule();
```

Algorithme 3 : creation for Yara rule

If we take the line 14 of this algorithm, it says "fileAAtraiter[i] is valid and with reading". Here, it's a checking to know if the line of the file to be treated corresponds to the standard that I detailed above, the size of the line, the checking by the name and the comparison with the result of *strings* of the basic machine or with that on the list of the software which is installed in the execution of the program.

To finish with this sub-section, it is important to talk about another point concerning the search for software with *grep* or during processing. During the project, one piece of software showed a problem, it is 7*zip*. As a reminder, the software is given to the client by a dictionary which will first have the name of the installer, then the name of the file *exe* for its execution and extraction from the machine. Only, in the case of 7zip the name of the *exe* file is 7z, the problem becomes clearer, because in a hard disk the readable character strings containing 7z are very numerous. It is following this problem that I added an additional step at the beginning of the program to make a *strings* on the software which will be installed, otherwise the execution is much too long, however it is not the point which I want to approach here, since even by reducing the execution time, the rules did not have much interest. Faced with this, I set up an intermediate list that I called blockProg, the name is not quite appropriate, but its purpose is to make the program understand that the name of the *exe* file is not the one that should be used during the search. With 7zip, we add a line in the list, "7z:7zip", simple but effective.

5.3.5 Yara rule for executable file

When running the client on the Windows virtual machine, the *exe* file of the software is extracted in order to create a Yara rule, so we need to see how these rules are created.

In order to be able to create a rule for the executable file, information about the file needs to be retrieved and hexadecimal data will be chosen. At the beginning, I selected the header and the end of a file *exe*, to make a rule, with the tool *xxd*. I don't know if several *exe* could have had the same rule but there was a change in the information retrieved quite quickly mainly because the rule created against the file would only work on it. What I have just said may seem paradoxical with what was said earlier in this paragraph, it is necessary to have a unique rule for the software and not only for the executable file, otherwise at each version change you have to create a new rule. Thus, it is necessary to take other data to answer the criteria, which will be the following: $CompanyName$, $ProductName$, $FileDescription$, $InternalName$, $OriginalFileName$. These data will in principle be unique to a software but always the same for each version, without forgetting that the same company can make several software but which will not have the same values for $FileDescription$ or $InternalName$ or $OriginalFileName$.

To retrieve this set, I use $pefile$ a python library which is dedicated to the *exe* file to be able to access some information like the one we are interested in here. I first used the library without including it in a code to understand the display and the part that is to be retained for the project, and in a second time I included it in a code to be able to automate the data recovery. The idea is to use $pefile$ to get the starting offset of this data which is the offset of $CompanyName$ and the ending offset which is the beginning of $FileVersion$ but which we are not interested in, the offset being the memory address to access it. Once everything is retrieved, I use *xxd* to get the hexadecimal code with *cut* to have only this part and not the readable ASCII part which is also displayed in the output of the command. The result of *xxd* will allow us to complete the second part of the Yara rule of the software, the *strings* part, and this will be the only thing to place, and finally the *condition* part, the third part of the Yara rule, will simply say that the one rule that is specified must be found.

This algorithm summarises the few instructions that need to be done:

```
1 Search for $CompanyName$;
2 Get the offset in Hexa from $CompagnyName$;
3 Get the offset in Hexa from $FileVersion$;
4 Length between $CompagnyName$ and $FileVersion$;
5 xxd and cut to retrieve the Hexa part of $CompanyName$, $ProductName$,
    $FileDescription$, $InternalName$ and $OriginalFileName$;
```

Algorithme 4 : Executable data recovery

This rule will apply to the disk directly, like the disk rules that were detailed in the sub-section above, and as only the executable will have this hexadecimal letter sequence, if it is not present it will not be detected.

5.3.6 Multi-logiciel

Towards the end of the project and the internship, Mr. Dulaunoy asked me to do something I had never done before, to install a software that will have several sub-software, in this case it is Libre Office.

Libre Office will have several subroutines like $Writer$ or $Calc$, similar to $Word$ and $Excel$ under Microsoft's office suite. So far I have only installed one main program each time, this is not expected, it requires some retouch in the program.

Firstly, the places that need to be changed to make this new constraint work must be identified, then the impact on the rest of the code must be measured, if the whole thing needs to be changed then the code is not well written. In our case, modifications are necessarily necessary but not excessively. Adding a new list is inevitable, a bit like the blockProg list which gives another name than that of the executable, you have to change the name of the executable to a list of executable which themselves The same can have an equivalent in blockProg, for example with LibreOffice: $LibreOffice.msi : soffice$, The basic list for the tasks to be carried out, the Multisoft list, which is the list to be added to our program, will be: $soffice : swriter, scalc...$ and finally blockProg: $swriter : Writer$, $scal : Calc...$ In the code, the *strings* and fls commands need to change to take into account the fact that more than one piece of software can be searched, which will create the results of fls and *strings* for each piece of software, no changes are needed in the processing of these files as they are already individual and will be processed in this way, changes would have been made if a single file encompassed all the different software. All the software is saved on the disk in the global folder of the software, for example for LibreOffice which will be the name of the folder on the disk which will contain the rules for $Writer$, $Calc...$

Finally, this algorithm summarises the main program developed, namely *Generator*:

```
Counting numbers of entries;
strings to optimise performance;
pour i from 0 to input number * 2 faire
    Write task to client;
    Launching the Windows machine;
    Conversion of the Windows machine;
    Retrieval fls;
    Retrieval strings|grep;
    si installation phase alors
        si AsaReport exists alors
            Treat AsaReport;
        fin
    fin
    Delete conversion;
fin
YaraRule();
Hashlookup();
```

Algorithme 5 : Generator

6 The future of the Project and its problems

Since the beginning of the project, improvements have been made continuously following requests from Mr Dulaunoy most of the time, so I know that there is always room for improvement and I also know that problems can still be found but also solved, so I will detail some future views that I think are feasible.

The program can be optimised to reduce the load and execution time mainly by modifying the *strings* command which takes the longest to run, about 30 minutes. I tried to reduce it by different methods such as using the *dd* tool to limit the disk partitions that will be read but the time was not reduced. In the same spirit, *ASA* (Attack Surface Analyzer), which runs on the client in the Windows virtual machine, takes a long time to process, there may be software, tools, solutions that could replace it, the execution time is about 30 minutes for the first collection, then about seven minutes for the second, the export takes a maximum of one minute. For the installation phase, with only two commands, more than an hour is used.

The processing to create the Yara rules is not the best, it works and gives results, but changes can surely make it much better in the results. Checking by name is still a rather basic way, but artificial intelligence would make it more advanced and with more complexity of understanding by reading it back. The different lists proposed like *blockProg* or *Multisoft* or even the list of tasks given to the client are also not very advanced and can surely be improved to be able to support changes easily in case of improvement of the global code. I would like to point out that I have tried as much as possible to make the code easy to understand and to do the job as well as possible.

A potential problem could be the understanding of the file to be filled in by the user, a file which serves to group together parameters necessary for execution which will be used in several programs but which, thus placed, are written and specified only once. I have of course added a description for each parameter and sometimes even examples to be as clear as possible.

One interesting addition would be that after identifying the installed software, the list of its vulnerabilities would be visible in some way, allowing the investigator using the rules to know whether a piece of software might have been targeted in an attack by giving him the list of possible vulnerabilities.

The use of *Hashlookup* could be greatly improved, currently the hashes are tested with this tool but no use is made of it afterwards, however it would be possible to make the comparison between the various results of *Hashlookup* on several softwares to determine which would be the entries specific to the system if ever they would be present of several softwares, which would make it possible to refine the exploitable results and to have only what is necessary, only what really corresponds to the installed software.

All the Yara rules created have in their conditions a variable that must be specified by the user to allow him to choose as he wishes these rules. It would be useful to be able to advise the user on what value to choose in order to get the best possible result when applying the rule. I have already written a small program to do this but it is not included in the main program and it is very time consuming to use as it takes at least 15 minutes to apply a disk rule.

Finally, a simple but effective improvement, especially to reduce execution times, is to have a high-performance machine, knowing that the remote server provided was already very efficient, much more so than my local machine.

7 Conclusion

Each major part of the code has been presented and explained, allowing to understand my choices in its implementation, algorithms have been written to facilitate the understanding of the explanations in addition to being used as a summary. A methodology has been presented, explained and used, knowing that it follows the main principles of investigation found in books dealing with the subject of Forensic* analysis or incident response. This set of key steps that I have been able to describe, which will have taken varying amounts of time to complete, have made it possible to achieve the goal set at the outset: to use remarkable artefacts to determine whether software has been installed or is installed on a machine. The goal has been achieved but more features have already been added and can be added as there is still a whole list of improvements available, all of which can be achieved more or less quickly.

This internship taught me a lot, both personally and professionally. It was only my second experience in a company but I felt integrated very quickly, even though most of the internship took place at a distance. The learning was not only in the field of work, the professional contact is one of the things you learn in a company, but also maturity in work management.

My university education was useful mainly in the development part of the project. Concerning the Forensic* analysis part, I learned the basics during my short training and before the internship with some readings that I did. I would have liked to have had some courses on the subject of Forensic* analysis in a more global way as we had some live memory analysis during our second year of the master, but it was not useful during the almost six months I spent on the internship.

To conclude this report, I would like to thank once again Alexandre Dulaunoy, my tutor at SMILE, for the time spent in his presence and for having given me the chance to work with him.

8 Glossaire

Economic Interest Group (GIE) : a grouping with legal personality which allows its members (of which there must be at least two) to pool some of their activities in order to develop, improve or increase the results of their activities while retaining their individuality. [12].

SME : Company with less than 500 employees.

Artefact : Objects that have legal value, i.e. all objects that contain data or evidence of something that has happened [13].

Forensic : A method of investigating crimes by examining evidence or artefacts.

Threat Intelligence : Discipline based on intelligence techniques, which aims to collect and organize all information related to cyber-attack threats.

SSH : Secure Shell is both a computer program and a secure communication protocol [14].

Registry : The registry (RDB) is a database used by the Windows operating system. It contains the configuration data of the operating system and other installed software wishing to use it. [15].

VirtualBox : Open source virtualisation software published by Oracle.

References

[1] https://github.com/CIRCL/factual-rules.

[2] https://github.com/CIRCL/factual-rules-generator.

[3] https://www.circl.lu/services/forensic-training-materials/.

[4] https://github.com/DavidCruciani/NoteGit/blob/main/README.md.

[5] https://github.com/DavidCruciani/NoteGit/blob/main/Location%20Artefact.md.

[6] https://www.oracle.com/technical-resources/articles/it-infrastructure/admin-manage-vbox-cli.html.

[7] https://community.chocolatey.org/packages.

[8] https://docs.microsoft.com/en-us/sysinternals/downloads/sync.

[9] https://docs.microsoft.com/en-us/sysinternals/downloads/sdelete.

[10] https://www.circl.lu/services/hashlookup/.

[11] https://github.com/microsoft/AttackSurfaceAnalyzer.

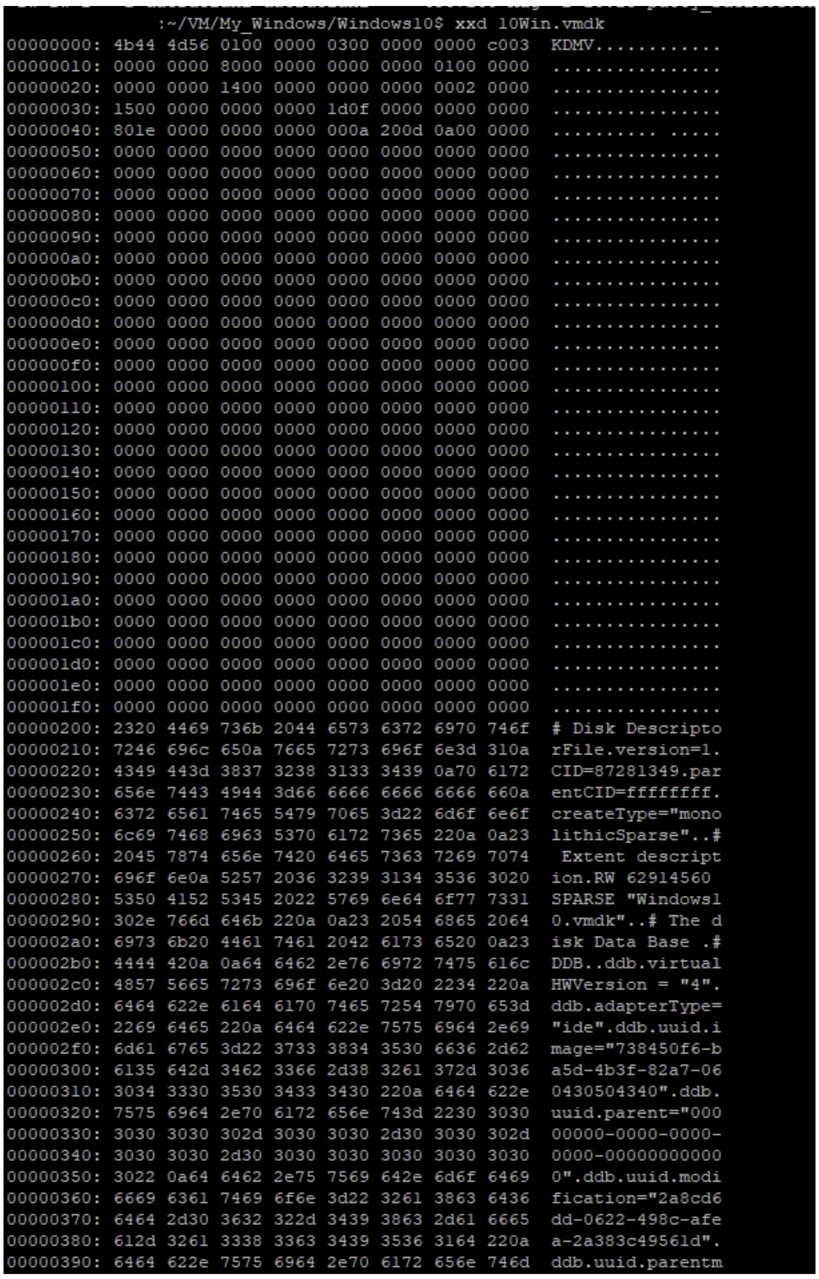
[12] https://fr.wikipedia.org/wiki/Groupement_d%27int%C3%A9r%C3%AAt_%C3%A9conomique.

[13] https://nasbench.medium.com/windows-forensics-analysis-windows-artifacts-part-i-c7ad81ada16c.

[14] https://fr.wikipedia.org/wiki/Secure_Shell.

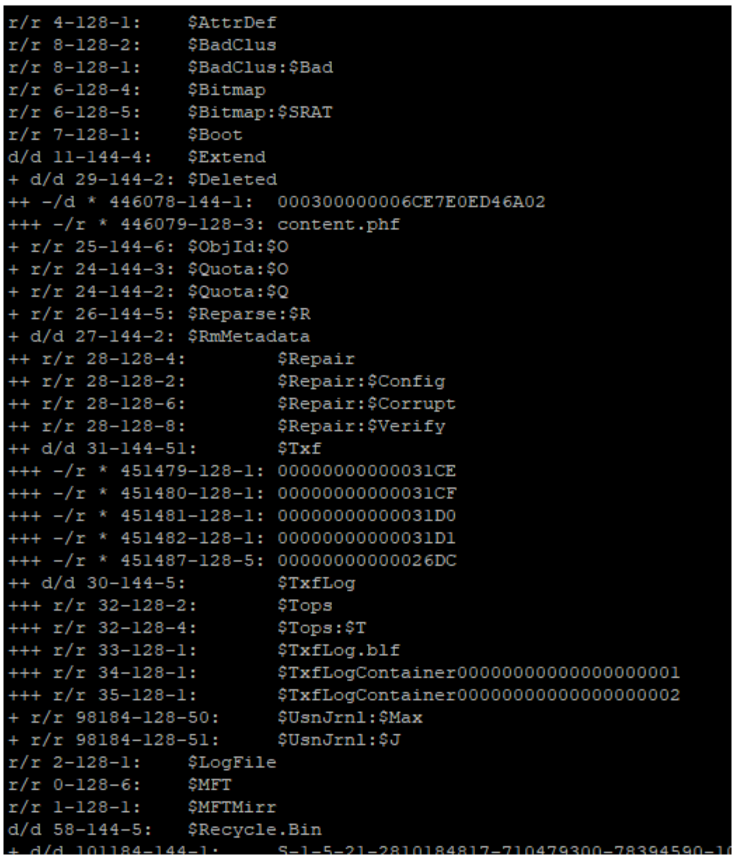
[15] https://fr.wikipedia.org/wiki/Base_de_registre.

9 Appendix



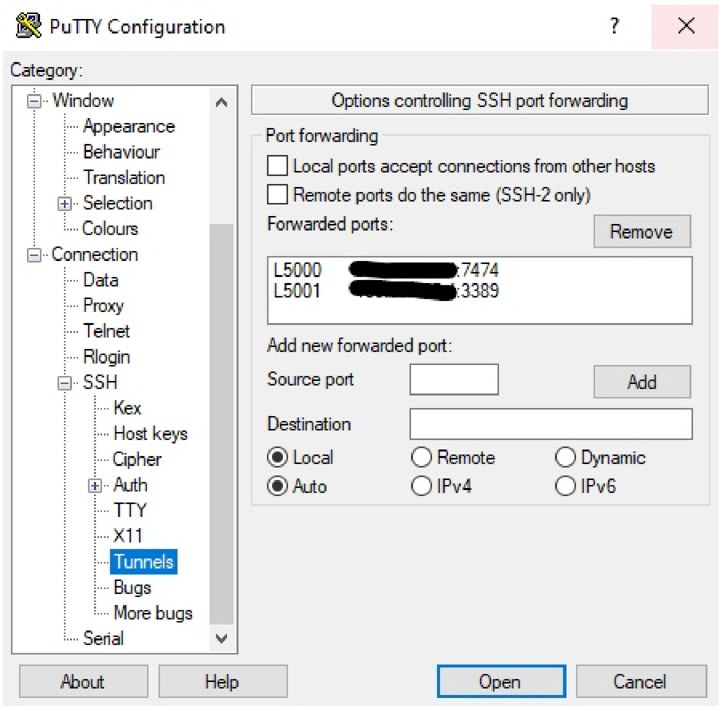
```
              :~/VM/My_Windows/Windows10$ xxd 10Win.vmdk
00000000: 4b44 4d56 0100 0000 0300 0000 0000 c003  KDMV............
00000010: 0000 0000 8000 0000 0000 0000 0100 0000  ................
00000020: 0000 0000 1400 0000 0000 0000 0002 0000  ................
00000030: 1500 0000 0000 0000 1d0f 0000 0000 0000  ................
00000040: 801e 0000 0000 0000 000a 200d 0a00 0000  .......... .....
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000160: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000180: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000190: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000200: 2320 4469 736b 2044 6573 6372 6970 746f  # Disk Descripto
00000210: 7246 696c 650a 7665 7273 696f 6e3d 310a  rFile.version=1.
00000220: 4349 443d 3837 3238 3133 3439 0a70 6172  CID=87281349.par
00000230: 656e 7443 4944 3d66 6666 6666 6666 660a  entCID=ffffffff.
00000240: 6372 6561 7465 5479 7065 3d22 6d6f 6e6f  createType="mono
00000250: 6c69 7468 6963 5370 6172 7365 220a 0a23  lithicSparse"..#
00000260: 2045 7874 656e 7420 6465 7363 7269 7074   Extent descript
00000270: 696f 6e0a 5257 2036 3239 3134 3536 3020  ion.RW 62914560
00000280: 5350 4152 5345 2022 5769 6e64 6f77 7331  SPARSE "Windows1
00000290: 302e 766d 646b 220a 0a23 2054 6865 2064  0.vmdk"..# The d
000002a0: 6973 6b20 4461 7461 2042 6173 6520 0a23  isk Data Base .#
000002b0: 4444 420a 0a64 6462 2e76 6972 7475 616c  DDB..ddb.virtual
000002c0: 4857 5665 7273 696f 6e20 3d20 2234 220a  HWVersion = "4".
000002d0: 6464 622e 6164 6170 7465 7254 7970 653d  ddb.adapterType=
000002e0: 2269 6465 220a 6464 622e 7575 6964 2e69  "ide".ddb.uuid.i
000002f0: 6d61 6765 3d22 3733 3834 3530 6636 2d62  mage="738450f6-b
00000300: 6135 642d 3462 3366 2d38 3261 372d 3036  a5d-4b3f-82a7-06
00000310: 3034 3330 3530 3433 3430 220a 6464 622e  0430504340".ddb.
00000320: 7575 6964 2e70 6172 656e 743d 2230 3030  uuid.parent="000
00000330: 3030 3030 302d 3030 3030 2d30 3030 302d  00000-0000-0000-
00000340: 3030 3030 2d30 3030 3030 3030 3030 3030  0000-00000000000
00000350: 3022 0a64 6462 2e75 7569 642e 6d6f 6469  0".ddb.uuid.modi
00000360: 6669 6361 7469 6f6e 3d22 3261 3863 6436  fication="2a8cd6
00000370: 6464 2d30 3632 322d 3439 3863 2d61 6665  dd-0622-498c-afe
00000380: 612d 3261 3338 3363 3439 3536 3164 220a  a-2a383c49561d".
00000390: 6464 622e 7575 6964 2e70 6172 656e 746d  ddb.uuid.parentm
```

Figure 1: *xxd* result



```
r/r 4-128-1:    $AttrDef
r/r 8-128-2:    $BadClus
r/r 8-128-1:    $BadClus:$Bad
r/r 6-128-4:    $Bitmap
r/r 6-128-5:    $Bitmap:$SRAT
r/r 7-128-1:    $Boot
d/d 11-144-4:   $Extend
+ d/d 29-144-2: $Deleted
++ -/d * 446078-144-1:  000300000006CE7E0ED46A02
+++ -/r * 446079-128-3: content.phf
+ r/r 25-144-6: $ObjId:$O
+ r/r 24-144-3: $Quota:$O
+ r/r 24-144-2: $Quota:$Q
+ r/r 26-144-5: $Reparse:$R
+ d/d 27-144-2: $RmMetadata
++ r/r 28-128-4:        $Repair
++ r/r 28-128-2:        $Repair:$Config
++ r/r 28-128-6:        $Repair:$Corrupt
++ r/r 28-128-8:        $Repair:$Verify
++ d/d 31-144-51:       $Txf
+++ -/r * 451479-128-1: 00000000000031CE
+++ -/r * 451480-128-1: 00000000000031CF
+++ -/r * 451481-128-1: 00000000000031D0
+++ -/r * 451482-128-1: 00000000000031D1
+++ -/r * 451487-128-5: 00000000000026DC
++ d/d 30-144-5:        $TxfLog
+++ r/r 32-128-2:       $Tops
+++ r/r 32-128-4:       $Tops:$T
+++ r/r 33-128-1:       $TxfLog.blf
+++ r/r 34-128-1:       $TxfLogContainer00000000000000000001
+++ r/r 35-128-1:       $TxfLogContainer00000000000000000002
+ r/r 98184-128-50:     $UsnJrnl:$Max
+ r/r 98184-128-51:     $UsnJrnl:$J
r/r 2-128-1:    $LogFile
r/r 0-128-6:    $MFT
r/r 1-128-1:    $MFTMirr
d/d 58-144-5:   $Recycle.Bin
+ d/d 101184-144-1:     S-1-5-21-2810184817-710479300-78394590-1
```

Figure 2: *fls* result



Options controlling SSH port forwarding

Port forwarding

☐ Local ports accept connections from other hosts

☐ Remote ports do the same (SSH-2 only)

Forwarded ports:

L5000 [redacted]:7474
L5001 [redacted]:3389

Add new forwarded port:

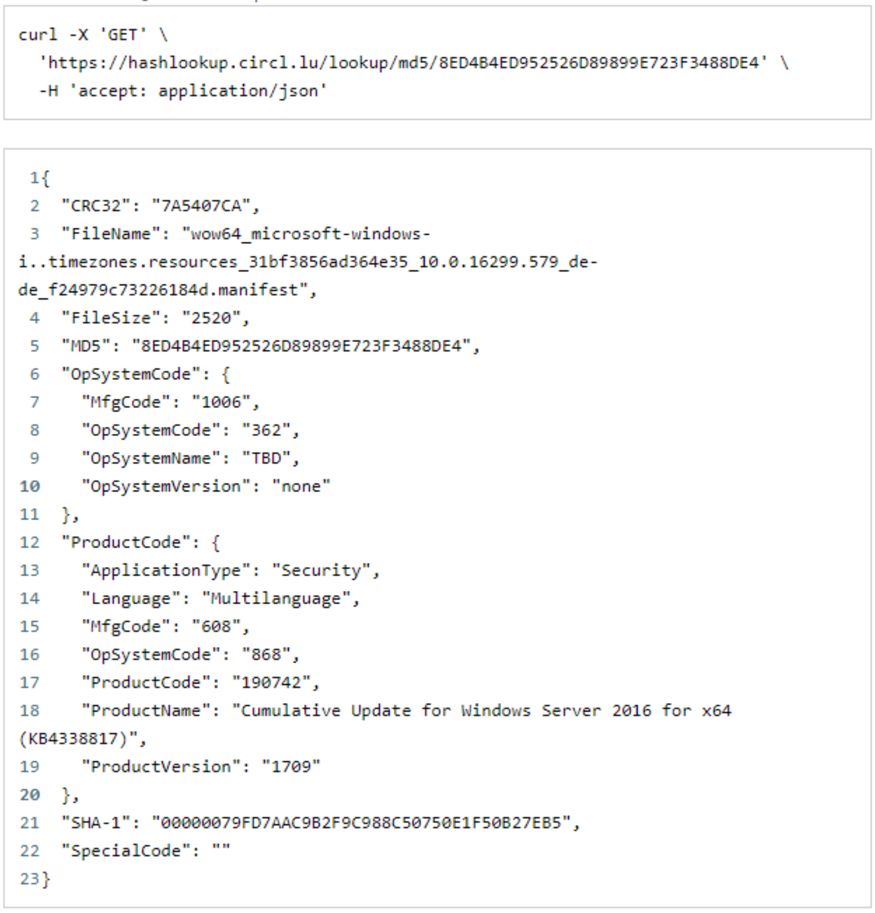
Source port

Destination

[✓ Local] Remote Dynamic

[✓ Auto] IPv4 IPv6

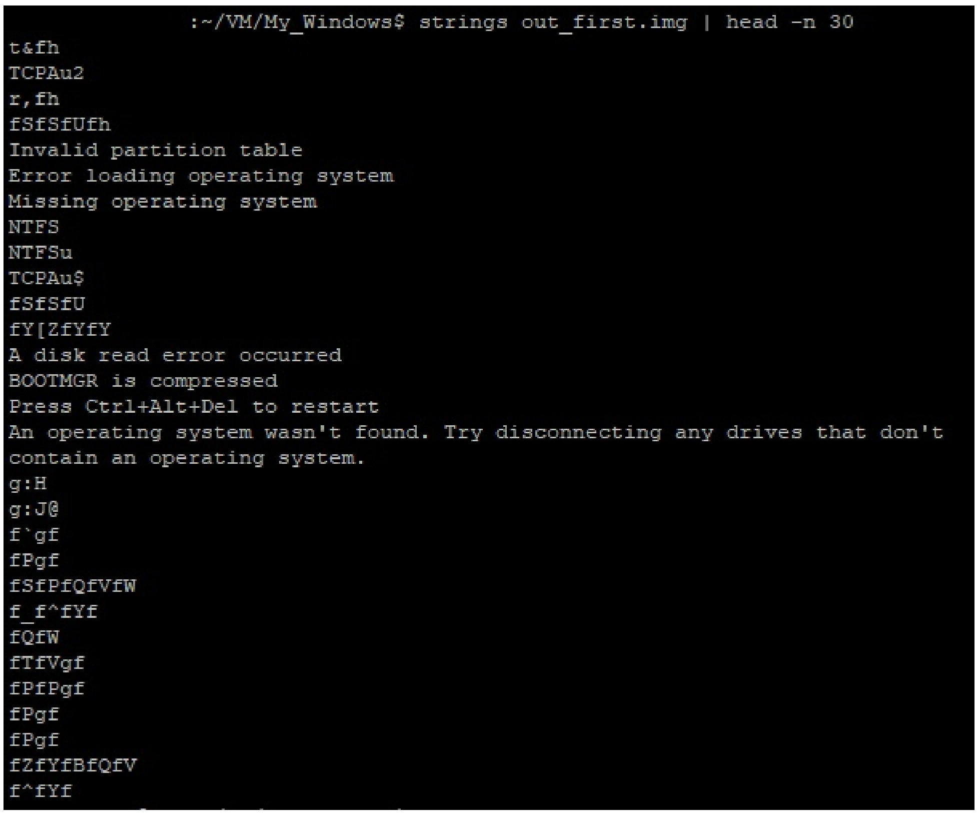
Figure 3: *PuTTY* tunnel



```
curl -X 'GET' \
  'https://hashlookup.circl.lu/lookup/md5/8ED4B4ED952526D89899E723F3488DE4' \
  -H 'accept: application/json'
```

```
{
  "CRC32": "7A5407CA",
  "FileName": "wow64_microsoft-windows-i..timezones.resources_31bf3856ad364e35_10.0.16299.579_de-de_f24979c73226184d.manifest",
  "FileSize": "2520",
  "MD5": "8ED4B4ED952526D89899E723F3488DE4",
  "OpSystemCode": {
    "MfgCode": "1006",
    "OpSystemCode": "362",
    "OpSystemName": "TBD",
    "OpSystemVersion": "none"
  },
  "ProductCode": {
    "ApplicationType": "Security",
    "Language": "Multilanguage",
    "MfgCode": "608",
    "OpSystemCode": "868",
    "ProductCode": "190742",
    "ProductName": "Cumulative Update for Windows Server 2016 for x64 (KB4338817)",
    "ProductVersion": "1709"
  },
  "SHA-1": "00000079FD7AAC9B2F9C988C50750E1F50B27EB5",
  "SpecialCode": ""
}
```

Figure 4: *Hashlookup* result



```
              :~/VM/My_Windows$ strings out_first.img | head -n 30
t&fh
TCPAu2
r,fh
fSfSfUfh
Invalid partition table
Error loading operating system
Missing operating system
NTFS
NTFSu
TCPAu$
fSfSfU
fY[ZfYfY
A disk read error occurred
BOOTMGR is compressed
Press Ctrl+Alt+Del to restart
An operating system wasn't found. Try disconnecting any drives that don't
contain an operating system.
g:H
g:J@
f`gf
fPgf
fSfPfQfVfW
f_f^fYf
fQfW
fTfVgf
fPfPgf
fPgf
fPgf
fZfYfBfQfV
f^fYf
```

Figure 5: *strings* result