Discriminating Unknown Software Using Distance Model

Yassine Lemmou, Hélène Le Bouder, Jean-Louis Lanet

Discriminating Unknown Software Using Distance Model

Yassine Lemmou
LABMIA
University Mohammed V
Avenue Ibn Batoufa, Rabat, Morocco
yassine.lemmou@gmail.com

Hélène Le-Bouder
IMT-Atlantique
2 Rue de la Chataigneraie,
35510 Cesson Cevigne, France
helene.le-bouder@imt-atlantique.fr

Jean-Louis Lanet
LHS INRIA-RBA
263 Avenue Gnral Leclerc,
35000 Rennes, France
jean-louis.lanet@inria.fr

***Abstract*—Crypto-ransomware is a class of malware that encrypt their victim's data and only return the decryption key in exchange for a ransom. In a previous work, we have yet designed a solution able to detect any ciphering of files using statistical estimator. Once detected, a pop up requests the user to verify if that operation is allowed on not. To improve our tool, automation is needed. In this paper, an anomaly detection mechanism to determine if a suspected group of threads is an authorized cryptographic software or a malicious code is presented. The effectiveness of our solution to correctly distinguish between valid programs and ransomware is evaluated using a string analysis. The `tf-idf` metric is used to choose the most pertinent features. The distance of a candidate software with a vector representing the allowed cryptographic software is measured. If the distance exceeds a threshold, the suspected process is flagged as a ransomware. We have evaluated our approach with the samples provided by open databases and executed on our bare metal platform.**

***Index Terms*—Ransomware, Machine Learning, Anomaly Detection, String analysis, TF-IDF metric**

I. Introduction

Ransomware are still one of the most dangerous security threats on the Internet. The growing number of ransomware attacks resulted in an increased concern on existent defense mechanisms. These malware share a similar objective: to render your data unavailable using encryption until you pay a ransom. It is of a prime importance to detect rapidly such an attack to restrain file losses if no prior detection was achievable. Current detection mechanisms rely on limiting this number of lost files. Immediately after the start of the encryption, they block any process that has similar behavior/features of a ransomware (Application Programming Interface (API) calls, registry keys, embedded strings in binaries...). Our solution does not rely on these features but only on data transformation. Thus leading to a solution independant of the code able to detect any new threat. A difficulty results in an approximate decision procedure with issues related with False Positive (FP) when a legitimate process is considered as a ransomware, and False Negative (FN) when a ransomware is detected as legitimate software. We have to rely on approximative decision procedures with False Positive (FP) and False Negative (FN) issues.

This paper improves our first countermeasure presented in [1], which targets crypto-ransomware. The main idea is that encrypted data is indistinguishable from perfectly random data. That is why our previous solution is implemented as a file system mini-filter driver. Statistical tool to detect random data are used.

We evaluate dynamically the entropy of a file with a Khi square statistical estimator or a divergence to a Markov chain's model while writing in the output buffer which represents our primary indicator. Once the driver detects a suspicious behavior, it blocks the process' threads and it performs a memory snapshot. It consists into the memory allocated to a process that is copied in another memory region not accessible to the suspected thread. Then, a pop-up warns the user of the suspected behavior. To automate this last feature, we need to distinguish between legitimate cryptographic applications (browser, backup, compressor etc.) and a ransomware which generates FP issues. The issue with an AI based solution remains FP and FN that have to be minimized. Currently, the driver performs well in term of FN (only 2 ransomware bypass the statistical test) but not with FP. We need to add a new detector, based on other features. The idea is to use the memory snapshot to check whether the suspected file is malicious or benign modeling the allowed software and then considering any deviation to this model as a threat.

The aim of this paper is to demonstrate that these snapshots can be used to extract relevant information such that we can classify the suspected thread. The driver transforms the memory snapshot into a binary vector that marks the presence or the absence of specific strings. Then, it verifies if the memory snapshot corresponds to the allowed cryptographic tools. It uses a similarity metric by measuring the cosine distance between the suspected snapshot and a model representing all the allowed cryptographic applications on that particular device. If the distance is close to one, the system releases the locks and lets the process carry on. If it is below a threshold, the system kills the threads and backup all the modified files.

The paper is organized as follows. The section II presents the context of the ransomware phenomenon and the state of the art of the countermeasure. In section III, the specific and technical context of our works is detailed. The new tool is presented in section IV. Experimental are described in

section V. Finally, the conclusion is drawn in section VI.

II. Ransomware story

A. The ransomware threat

The first occurrence of a ransomware was in 1989 [2]. The pandemic began in 2012, when the number of ransomware victims has increased significantly. The crypto-currency made the success of these malware such as bitcoin, for the trade which is nearly impossible to trace. The ransomware business model works, people pay, and thus conducts to an augmentation of 266 percent of the average ransom compared to a year ago (*i.e.*, 1077$) [3]. Ransomware as a service is growing, the Cerber authors hold an affiliate program and get back money for each franchised infection when the victim pay [4]. Indeed, it is only the beginnings of the ransomware surge. The worldwide `wannaCry` attack emerged by exploiting the MS17-010 flaw. A ransomware attack can be split in separated step. The vectors for ransomware infection are classical : trapped e-mail, or usb-key ....

Generation of symmetric keys: Generally a ransomware uses symmetric encryption as AES to encrypt data. First step is to generate symmetric keys, with a random generator. One symmetric key is associated at one victim's file.

Then a public key is used to encrypt symmetric keys. Sometime the public key is embedded with the code.

Foot-printing the target: The ransomware checks if the system has not already been encrypted, search for files to be encrypted. It needs to perform a complete file traversal, they mostly use Windows API to find attached removable devices, and browse the file system. The ransomware browses files to find specific victims (.doc, .xls, .jpg, .pdf, .mp3)

Encryption of data: A ransomware encrypts files with a symmetric cipher. The secret key is ciphered using the public key and is added to the file. Some ransomware use the crypto API of Windows while others use their own embedded cryptography.

Ransom note: A ransom note is displayed to explain how to pay to the victim. This marker must be significantly visible being part of the business model. Finally if the victim agrees to pay, the ransomware rescue the files by handing the symmetric keys. In practice, it is often the case but sometimes the attacker do not restore data.

The dangerous behavior is of course the third one, all the other behaviors can be used to confirm or infirm the result of the countermeasure or as an early warning.

B. State of the art of the countermeasures

One of the most used technique is signature-based method. It is a sequence of information that characterizes each known malware. The detection is generally a two-step process: feature extraction and classification/clustering. In the first step, various features such as API calls, binary strings, and program n-gram are extracted statically and/or dynamically to capture the characteristics of the file samples. In the second step, techniques such as classification or clustering are used to automatically categorize file samples into different classes based on the analysis of the feature representations. These data-mining-based malware detectors mainly differ on the feature representation and the employed data mining techniques. The features can be Windows API calls [5], byte n-grams [6], strings [7], instructions [8], and control flow graphs [9].

The authors of [5] apply data mining methods on a dataset of malicious executables where a set of Windows and MS-DOS format executables are utilized. They used strings and 2-byte sequences as features. They obtained very high recall and precision with 5-Fold Cross Validation on a dataset of 4 266 Windows executables (3 265 known malicious binaries and 1 001 benign). A rule induction algorithm Ripper [10] was applied to discover patterns based on DLL calls dataset. The authors conclude that malware detection rate using data mining method was twice of the signature-based method on their data collection.

In [8] the authors develop a system for Android where they extracted all the strings from the disassembled files of Android binaries. They transform the text into vectors using the `TF-IDF` weighting schema. No indication is given on the size of the vector. Then, they compute the distance between the file under investigation and a vector representing benign software.

Simply creating signatures using the call frequency does not allow them to detect malware in polymorphic or unknown form. It can be also evaded by malware authors' by inserting redundant API calls. Several authors use sequences of system calls, API calls and function calls of malware to detect malicious behaviors. In [11], they use sequence of function calls to represent program behavior.

All these works try to identify a malware by searching a set of strings to flag it as hostile knowing a set of signatures. We do not rely on string analysis to flag it as hostile, we rely on data modification for that. We use string analysis only to eradicate the false positive. The model of the strings does not use the binary code of the malware but on the binary of the authorized encrypting software.

III. Context

A. The bare metal evaluation platform

An automated analysis platform called MoM (Malware O Matic) has been designed. It does not use a virtual machine but it maintains all the main features of a regular analysis framework. Such a fully bare-metal platform is built on top of two open source software, Clonezilla [12] and Viper [13], which makes it reusable. The platform is made of a master server and several slaves. Each slave runs the analysis loop in parallel. The whole system is on a dedicated network under the supervision of the master server and directly connected to the Internet, to emulate a typical home network. The malware database and the result of the analyses are stored on a Network Attached Storage (NAS). MoM grabs automatically new sample of ransomware from public repositories. Then, these samples are classified and for those that are alive (they have all the conditions to be executed) we evaluate our run time analyses. A bare metal platform is preferred to the

solutions based on virtualization in order to avoid well known evasion techniques. It exists numerous techniques used by the malware to fingerprint well-known sandboxes (*e.g.*, Cuckoo Sandbox [14]). If a sandbox is detected, they can evade them avoiding to deploy the payload.

The analysis loop configures the monitoring environment, executes the malware, gathers the results and performs a clean up of the system. In the first step, the slave downloads a script from the master, a set of instructions on how to conduct the next analysis. Once the procedure is completed, the slave sets its next environment and reboots for cleanup. The clean-up process consists in flashing a clean disk image into the slave's drive. Windows 10 and Seven, 32-bit or 64 bit disk image are used as the operating system to be infected. The user is logged in as administrator with the User Account Control (UAC) disabled. Each run is fifteen minutes long. After that period, if the ransomware did not deploy its payload, it is tagged as inactive and removed of the data base. Else, several data are collected for signature extraction, run time detection evaluation and for post mortem analysis.

We have included into the Windows images our mini-filter such that after each execution we can add to our database the memory snapshot. In the database, all the samples are associated with their memory snapshot.

B. Preliminary works

This paper is an improvement of a first countermeasure presented in [15]. The solution is implemented as a filter driver which can inspect all the I/O operations that target the disks. It does not matter that the requested operation be an I/O Request Packet (IRP) or a FAST I/O. In this context, we are able to monitor write, read operations, change directory and so on. However, a clever usage of such functionalities have to be done, otherwise a significant penalty occurs and the solution can not be deployed in real world. We restrict our monitoring on the write operations. Our solution has been extensively tested on Windows 7 and 10, whatever the architecture 32 or 64-bit.

To detect a ransomware statistical tool are used. The detector uses the property that the distribution of encrypted data is similar to random data. This solution is based on the Khi square which measures the frequency distribution. The chi-square goodness-of-fit test is a test of distributional accuracy, it measures how closely a set of numbers follows a particular distribution. It is a non-parametric statistical test, meaning that no assumption is done on the samples distribution. This test can distinguish randomness (or encryption) from some compression schemes and is the more relevant statistic distinguisher. This detector is particularly well suited to detect a file encryption. But it is unable to distinguish an authorized ciphering tool (OpenPGP) or compressor (WinZip) from a ransomware, thus generating false positive.

IV. Reducing the false positive

Once a thread has been detected to be suspicious, the driver takes a memory snapshot related to the process to which the thread belongs.

A. Strings and vectors

It uses a string analysis on the content of the memory snapshot to reduce the FP. The strings represent the `tokens` or the `features` in the different algorithms. Memory of a suspected file is made of code, data and garbage. A binary code calls functions belonging either on static or dynamic libraries (DLL). The strings corresponding to these items have to be present in memory while the code is running. The use of a dynamic bare metal approach, avoids us two problems: obfuscation and evasion. Memory is saved while the payload has been deployed in memory (even if drop byte effect can occur) and is active (ciphering has been detected). Each program (benign or malicious) can be represented by either by a boolean or an integer vector of the features. A boolean vector of the features indicates the presence or not of the different strings while an integer vector represents the raw count of the strings. We choose to use the integer representation for the vector c of the candidate program.

To check if a suspicious memory is malicious or not, we have to compare it with a model m representing a set of authorized ciphering software. Such a list has to be defined by the administrator of the device. He knows all the software installed within each machine including the ciphering software. For each software, the tool takes a memory snapshot and extract the strings. This model has been build with the allowed ciphering tools as `challenger, cryle, hamster, veracrypt, axcrypt, cryptoexpert, master voyage, cryptoforge`; archivers with ciphering capabilities as `peazip, 7zip, winzip, winrar` and a specific browser, `Tor`. Only intersection of all strings are used to build the model.

B. Memory filtering

When a memory snapshot is analyzed, the first step consists in removing useless information from the memory. A snapshot contains thousand of strings. Some of them are just noise (string with no semantics) as code section or previously allocated but uncleaned memory.

The memory can include the following items:

- Noise, code section with character like instructions,
- Ransom info, `e.g.`194.132.208.167 : 3114 197 e09caa5d3f7664be5c812ea6f1425dfd467e8, to e-mail address sukhonina.akilina@gmail.com,
- Execution data: `e.g.`the string advapi32.dll

This step consists in removing information that are useless for the detection. Two elements are of importance: the loaded DLL and the name of the functions called. To keep only the strings with relevant meaning, a dictionary approach (white list) is used. It reduces the number of strings by one order of magnitude keeping only relevant information. An efficient hash map structure is used for the dictionary. At that step, the strings are only valid strings. We can build the vector c counting their raw value in the memory. We build c representing a software in two parts: the DLL section and the function names

section. The DLL part is of a fixed length. It corresponds to all the allowed software encountered at that time. We select only a valuable part of the strings for the vector according to the amount of information it represents.

C. Compute the best vector

The second step consists in reducing the size of the vector. Reducing the vector is necessary because all the valid software do not have the same number of strings. If we expect to compare dynamically the vectors, we need to normalize its size. For that purpose, we can apply Information Retrieval (IR) techniques to select a subset of the strings in the vector. The set of strings of a vector can now be considered as a document and the set of valid application as a collection of documents.

We use the `tf-idf` weighting technique to choose the relevant strings to be kept in the vector. The value associated to each term has the following explanation:

- highest when the term occurs many times within a small number of documents leading to a high discriminating power;
- lower when the term occurs fewer times in a document, or occurs in many documents;
- is the lowest when the term occurs in almost all documents.

To compute the `tf-idf`, we first compute for each term of a document its `tf`. This value reflects the normalized occurrence of the term in the document. We keep only as useful information its frequency considering the order as irrelevant. In that representation, each term (function calls of the DLL) has the same importance. In fact, certain terms have little or no discriminating power in determining relevance. For example, the term `EnableWindow` which is present in the DLL is useless for discriminating goodware and malware. We compute the document frequency `df` defined to be the number of documents `N` in the collection that contain a given term `t` . Then, the inverse document frequency `idf` is computed as the $log(N/df_t)$. We combine now these two metrics to build the `tf-idf` value as the product of $tf_{t,d}$ x idf_t. Then, we keep only a subset of variables (function name) to take part of the vector of each authorized software. We observed that the minimum value is 25 variables, a good trade-off is between 100 and 400 variables and keeping the variables over 500 does not improve the results.

Any candidate c and then be compared to each of these vectors. We choose to use only one representation of these vectors by computing the barycenter of these vectors. It forms our model m of the allowed ciphering software.

D. Distance to the model

In this paragraph, the distance between a candidate vector c (the suspicious one) and the model m is evaluated. The vector c can be seen has the distribution of the strings in the memory. Various distance/similarity measures are available in literature to compare two data distributions. To measure the divergence, we evaluate two metrics, the euclidean and the cosine distance. The euclidean distance (1) between each element of the candidate vector c and the model m is used, assuming n being the size of the both vectors.

$$d(m,c) = \sum_{i=1}^{n} (m_i - c_i)^2 \tag{1}$$

It is the ordinary distance between two points. The Euclidean distance determines the root of square differences between the coordinates of a pair of objects.

The cosine distance (2) is used too, even if at the end only one metric will remain in the driver.

$$\cos(m,c) = \frac{mc}{\|m\|\|c\|} = \frac{\sum_{i=1}^{n} m_i c_i}{\sqrt{\sum_{i=1}^{n} {m_i}^2}\sqrt{\sum_{i=1}^{n} {c_i}^2}} \tag{2}$$

Setting up the vector consists in choosing the most interesting features to discriminate then. The more element of the vector we have the more precise the computation is but also the most costly. We will discuss in the section V the trade off we made.

V. Experimental results

A. Efficiency of the divergence model

In this paragraph, the evaluation of the ability of the model to reject illegal ciphering software is presented. First the minimum value of strings required for a good detection ratio from 50 values to 500 is defined experimentally. TABLE 1 shows the different values for a candidate `bitman`. Reducing the size of the vector reduces the time needed for dynamically comparing the two vectors. The smaller they are, quicker the on-line decision can be taken to kill or not the process.

All the allowed ciphering software are below 45 for the Euclidean distance and above 0.92 for the Cosine distance (they match the model). We setup the threshold of the Euclidean distance to 50 and the cosine distance 0.85 with 400 features. If a candidate reaches these thresholds, then it is labeled as an allowed ciphering tool.

	Euclidean	Cosine
50	44	.34
100	87	.36
150	131	.38
200	170	.38
250	220	.34
300	268	.32
350	318	.30
400	368	.28
450	418	.26
500	468	.26

TABLE I
MEAN VALUE OF THE DISTANCES ACCORDING TO THE NUMBER OF FEATURES

Several ransomware are used, samples of the following families: `bitman`, `cerberV1`, `cerberv2`, `fsnyna`, `telsacrypt`, `xorist`, `yakes`, `zerber`, `crysis`, `ctblocker`, `gamarue`, `graphtor`, `locky`, `purge` and `sage`.

The results for a threshold of 400 strings are given in Table 2. There is clear separation between the class of allowed ciphering tools and the ransomware.

	Euclidean $\geq$ 50	Cosine $\leq$0.85
bitman	368	.28
cerberV1	350	.35
cerberV2	2907	.06
fsnyna	341	.38
telsacrypt	368	.28
xorist	376	.24
yakes	365	.29
zerber	356	.33
crysis	399	.05
ctblocker	395	.11
gamarue	361	.31
graphtor	395	.11
locky	386	.18
purge	376	.24
sage	383	.20

TABLE II
DETECTION RATIO OF THE RANSOMWARE SET

If an allowed tool is upgraded to a new version, the model m must be recomputed. We evaluate the impact of new versions of the software and how it impacts the model. For this reason, we integrate in the model the version 15.5 of winzip (the 32 bit version) and we check the behavior of the detector if a user loads an upgraded version. We evaluated the versions: 16.5 (64 bits), 17.5 (64 bits) and 22.0 (64 bits). All of them have been detected as valid programs with an Euclidean distance close to 0 and a cosine close to 1.

B. Known Issues

The first issue is related to the false positives that this solution can raise. We evaluate many other programs (Office Powerpoint, Word, Excel, DiskMark, Firefox, Python, SearchIndexer, Open Office Calc...) and they would have been classified as malware with a value below 0.5 for the Cosine metric and over 350 for the Euclidean distance. But due to the fact that they do not perform encryption, they are not suspected by the primary indicator. This solution would not have been suitable if the preliminary step was not executed first.

The second issue is related to the strength of the solution. Any software can be able to bypass this indicator by inserting strings in its code. If the model is stored in the kernel like in our solution, then the attacker must guess the elements that are part of the model. We have restricted the number to 755 common strings, but if we use also the loaded DLL we can include much more common strings but at a higher cost for on line decision. If the attacker can fingerprint the system in a preliminary step, she knows the different installed software. She has to add all the strings in a data section of his malware to bypass the detector. Until now such a behavior has never been detected but it can occur.

C. Global efficiency

For the global evaluation of our solution, *i.e.* the primary indicator followed buy the string analysis, the following classical measures are used to evaluate its performance. The True Positive Rate (TPR) measure also called the sensitivity is the rate of ransomware samples (*i.e.*, positive instances) correctly recognized:

$$TPR = \frac{TP}{TP + FN} \quad ,$$

with TP the number of applications correctly classified as ransomware and FN the number of applications misclassified as benign software.

The False Positive Rate (FPR) is the rate of authorized files (*i.e.*, negative instances) wrongly classified (*i.e.*, misclassified as ransomware samples):

$$FPR = \frac{FP}{FN + TN} \quad ,$$

with FP the number of valid applications incorrectly detected as malicious and TN the number of valid applications correctly classified. The Accuracy (ACC) measure is the rate of the correctly classified file instances, including both positive and negative instances.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \quad .$$

The input set of data consists in two subsets. A benign set, which corresponds to a collection of intensive use of scientific laptop during nine days containing around 110 000 threads. A hostile set, which corresponds to the execution of ransomware of different families (365 threads).

In the benign set, we can find threads related to the automatic backup, the anti-virus solutions and some compressor tools. TABLE 3 provides the result corresponding to the detection using the first level indicator.

In the second column labeled Current work is the improvement obtained with this work. This solution acts only on the FPR by distinguishing the benign software that have been detected as suspected threads. It has no effect on the TPR metric. We consider that all the authorized software have been included into the divergence model. If this is not the case, FP increases correspondingly. The policy of authorized software must remain under the control of the administrator which can raise problem with individuals who are not aware of the dangerousness of software uploaded from unknown sources. This can be a limit of this approach if the user is the administrator.

Metric	Previous solution III-B	Current work
TP	365	365
TN	103 180	103 180
FP	6820	0
FN	15	15
TPR	96.05%	96.05%
FPR	17.9 %	0%
ACC	93,79%	99.98%

TABLE III
METRICS RELATED TO THE SOLUTION

VI. Conclusions

In this work, we develop an anomaly detection process of suspected cryptographic activities in threads to replace a human decision. We use a model for allowed cryptographic applications having a subset of common features. We build a model representing these allowed applications. We measure the distance between a suspected thread and the model to allow legitimate software to resume execution and to block the illegal ones. We improve the original solution by reducing the FP rate but with no effect on the FN. We mix several detectors, which use different features, that measure different aspects of an unknown software. The solution relies on a single probe that records all the writing requests. Its impact on the system is extremely low, the second indicator cost more but it is only involved in case of suspicion. As a drawback, this mechanism can be bypassed if the attacker can customize dynamically his code after a fingerprinting step.

Appendix

The complete list of the ransomware tested is available at the following address: `people.rennes.inria.fr/ Aurelien.Palisse/ ml-labels.txt` with their MD5. The list of allowed cyphering tools that have been used to form the model: `challenger`, `cryle`, `hamster`, `veracrypt`, `axcrypt`, `cryptoexpert`, `master voyage`, `cryptoforge`; archivers with ciphering capabilities as `peazip`, `7zip`, `winzip`, `winrar` and a specific browser, `Tor`.

References

[1] Aurelien Palisse, Antoine Durand, Helene Le Bouder, Colas Le Guernic, and Jean-Louis Lanet. *Data aware defense (dad): Towards a generic and practical ransomware countermeasure*. In Nordic Conference on Secure IT Systems, pages 192208. Springer, 2017.

[2] Adam L. Young and Moti Yung. *Cryptovirology: Extortion-based security threats and countermeasures*. In 1996 IEEE Symposium on Security and Privacy, May 6-8, 1996, Oakland,CA, USA, pages 129140, 1996.

[3] Patrick Howell ONeill. *Ransomware demands now average about 1,000 dollars because so many victims decide to pay up*, April 2017.

[4] Malwarebytes. *Cybercrime tactics and techniques*. Technical report, March 2017.

[5] [Matthew G Schultz, Eleazar Eskin, F Zadok, and Salvatore J Stolfo. *Data mining methods for detection of new malicious executables*. In Security and Privacy, 2001. Proceedings. 2001 IEEE Symposium on, pages 3849. IEEE, 2001.

[6] Mohammad M Masud, Latifur Khan, and Bhavani Thuraisingham. *A scalable multi-level feature extraction technique to detect malicious executables*. Information Systems Frontiers, 10(1):3345, 2008.

[7] Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. *Sbmds: an interpretable string based malware detection system using svm ensemble with bagging*. Journal in computer virology, 5(4):283, 2009.

[8] Igor Santos, Felix Brezo, Javier Nieves, Yoseba K Penya, Borja Sanz, Carlos Laorden, and Pablo G Bringas. *Idea: Opcode-sequence-based malware detection*. In International Symposium on Engineering Secure Software and Systems, pages 3543. Springer, 2010.

[9] Marius Gheorghescu. *An automated virus classification system*. In Virus bulletin conference, volume 2005, pages 294300. Citeseer, 2005.

[10] William W Cohen. *Learning trees and rules with set-valued features*. In AAAI/IAAI, Vol. 1, pages 709716, 1996.

[11] Sean Peisert, Matt Bishop, Sidney Karin, and Keith Marzullo. *Analysis of computer intrusions using sequences of function calls*. IEEE Transactions on dependable and secure computing, 4(2):137150, 2007.

[12] Clonezilla. *The Free and Open Source Software for Disk Imaging and Cloning*, v:2.5.6-21, 2018.

[13] Viper. *Binary management and analysis framework*, 2015.

[14] Cuckoo Foundation. *Cuckoo Sandbox 2.0.6: Automated Malware Analysis*, 2018.

[15] Aurelien Palisse. *Analyse et detection de logiciel de rancon*. PhD thesis, University of Rennes 1, 2019.