PhD-FTSM-2020-57
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Defence held on 14/10/2020 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Ziya Alper GENÇ
Born on 26 November 1987 in Adana, Turkey

Analysis, Detection, and Prevention of Cryptographic Ransomware

Analysis, Detection, and Prevention of Cryptographic Ransomware

Ziya Alper Genç

Faculty of Science, Technology and Medicine, 2 Avenue de l'Université, L-4365 Esch-sur-Alzette, Luxembourg.

Interdisciplinary Centre for Security, Reliability and Trust (SnT), 29, Avenue J. F. Kennedy, L-1855 Luxembourg.

The work in this thesis has been carried out under the auspices of the Interdisciplinary Research Group in Socio-technical Cybersecurity (IRiSC) of University of Luxembourg.

This research is partially supported by the Luxembourg National Research Fund (FNR), through the JUMP project grants PF18/12536861/NoCry and PoC18/13234766/NoCry.

This thesis was typeset with LuaTEX Version 1.12.0 (TEX Live 2020).

Publisher
University of Luxembourg Press, Esch-sur-Alzette, Luxembourg.

Declaration of Authorship

I, Ziya Alper GENÇ, declare that this thesis titled, "Analysis, Detection and Prevention of Cryptographic Ransomware" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at University of Luxembourg.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at University of Luxembourg or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Luxembourg, October 14, 2020

Ziya Alper GENÇ

Sevgili annem ve babam, Necla ve Ahmet'e,
canım kardeşim İsmail'e.

Babaannem Nuriye'nin aziz hatırasına.

To my beloved parents, Necla and Ahmet,
to my dear brother İsmail.

In memory of my grandmother Nuriye.

Abstract

Cryptographic ransomware encrypts files on a computer system, thereby blocks access to victim's data, until a ransom is paid. The quick return in revenue together with the practical difficulties in accurately tracking cryptocurrencies used by victims to perform the ransom payment, have made ransomware a preferred tool for cybercriminals. In addition, exploiting zero-day vulnerabilities found in Windows Operating Systems (OSs), the most widely used OS on desktop computers, has enabled ransomware to extend its threat and have detrimental effects at world-wide level. For instance, `WannaCry` and `NotPetya` have affected almost all countries, impacted organizations, and the latter alone caused damage which costs more than $10 billion.

In this thesis, we conduct a theoretical and experimental study on cryptographic ransomware. In the first part, we explore the anatomy of a ransomware, and in particular, analyze the key management strategies employed by notable families. We verify that for a long-term success, ransomware authors must acquire good random numbers to seed Key Derivation Functions (KDFs).

The second part of this thesis analyzes the security of the current anti-ransomware approaches, both in academic literature and real-world systems, with the aim to anticipate how such future generations of ransomware will work, and in order to start planning on how to stop them. We argue that among them, there will be some which will try to defeat current anti-ransomware; thus, we can speculate over their working principles by studying the weak points in the strategies that six of the most advanced anti-ransomware currently implements. We support our speculations with experiments, proving at the same time that those weak points are in fact vulnerabilities and that the future ransomware that we have imagined can be effective. Next, we analyze existing decoy strategies and discuss how they are effective in countering current ransomware by defining a set of metrics to measure their robustness. To demonstrate how ransomware can identify existing deception-based detection strategies, we implement a proof-of-concept decoy-aware ransomware that successfully bypasses decoys by using a decision engine with few rules. We also discuss existing issues in decoy-based strategies and propose practical solutions to mitigate them. Finally, we look for vulnerabilities in antivirus (AV) programs which are the *de facto* security tool installed at computers against cryptographic ransomware. In our experiments with 29 consumer-level AVs, we discovered two critilcal vulnerabilities. The first one consists in simulating mouse events to control AVs, namely to send them mouse "clicks" to deactivate their protection. We prove that 14 out of 29 AVs can be disabled in this way, and we call this class of attacks *Ghost Control*. The second one consists in controlling whitelisted applications, such as `Notepad`, by sending them keyboard events (such as "copy-and-paste") to perform malicious operations on behalf of the malware.

We prove that the anti-ransomware protection feature of AVs can be bypassed if we use `Notepad` as a “puppet” to rewrite the content of protected files as a ransomware would do. Playing with the words, and recalling the cat-and-mouse game, we call this class of attacks *Cut-and-Mouse*.

In the third part of the thesis, we propose a strategy to mitigate cryptographic ransomware attacks. Based on our insights from the first part of the thesis, we present UShallNotPass which works by controlling access to secure randomness sources, i.e., Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) Appliction Programming Interfaces (APIs). We tested UShallNotPass against 524 real-world ransomware samples, and observe that UShallNotPass stops 94% of them, including `WannaCry`, `Locky`, `CryptoLocker` and `CryptoWall`. Remarkably, it also nullifies `NotPetya`, the offspring of the family which so far has eluded all defenses. Next, we present NoCry, which shares the same defense strategy but implements an improved architecture. We show that NoCry is more secure (with components that are not vulnerable to known attacks), more effective (with less false negatives in the class of ransomware addressed) and more efficient (with minimal false positive rate and negligible overhead). To confirm that the new architecture works as expected, we tested NoCry against a new set of 747 ransomware samples, of which, NoCry could stop 97.1%, bringing its security and technological readiness to a higher level.

Finally, in the fourth part, we present the potential future of the cryptographic ransomware. We identify new possible ransomware targets inspired by the cybersecurity incidents occurred in real-world scenarios. In this respect, we described possible threats that ransomware may pose by targeting critical domains, such as the Internet of Things and the Socio-Technical systems, which will worrisomely amplify the effectiveness of ransomware attacks. Next, we looked into whether ransomware authors re-use the work of others, available at public platforms and repositories, and produce insecure code (which might enable to build decryptors). By methodically reverse-engineering malware executables, we have found that, out of 21 ransomware samples, 9 contain copy-paste code from public resources. From this fact, we recall critical cases of code disclosure in the recent history of ransomware and, reflect on the dual-use nature of this research by arguing that ransomware are components in cyber-weapons. We conclude by discussing the benefits and limits of using cyber-intelligence and counter-intelligence strategies that could be used against this threat.

Acknowledgements

First, I would like to thank my supervisor Gabriele Lenzini of the Interdisciplinary Research Group in Socio-technical Cybersecurity (IRiSC) of University of Luxembourg. Without Gabriele's constant guidance and encouragement, this thesis would not be possible. In addition to his scientific capabilities and constructive criticism which have empowered me to produce high quality works, Gabriele is one of the kindest supervisors one could ever have. After four years of study, I feel that it is a great honor for me to be a PhD student of Gabriele. I would also like to thank another scientist who greatly inspired me, Peter Ryan of Applied Security and Information Assurance Group (APSIA), for providing insightful feedback and contributing to valuable discussions regarding my research. It has been a privilege to work in these two vivid research groups, APSIA and IRiSC, where the abstract thoughts find their true value.

I thank Daniele Sgandurra, who collaborated to my research (three chapters of this thesis); Sjouke Mauw, who dedicated his time to evaluate my progress over these years; Jean-Louis Lanet, and Gianluca Stringhini, who were kind enough to accept being members of my dissertation committee. I appreciate all your efforts.

I also thank our administrative assistants Ida Ienna and Natalie Kirf whose strong organizational skills have made my (indeed everyone's) life much easier in our research group.

A very special thanks goes to Birnur Daultrey for being an invaluable friend. I must note that, Birnur is an always-positive person and a true master of "energy". For a long term project, like this PhD thesis, it is essential to keep up the motivation. Thanks to Birnur, it has never been a problem. I also like to thank a talented musician, Simon Daultrey: I still remember the video clips of your guitar solos. Thank you both, it is a great chance for me to have friends like you.

I also thank İsa Sertkaya for his friendship and for reading a chapter of my thesis. Our fruitful discussions on various topics have lead to new areas to be explored, but the list is probably too long for a short life.*

This research was also made possible with the support received from the Luxembourg National Research Fund (FNR), through the JUMP project grants PF18/12536861/NoCry and PoC18/13234766/NoCry. I would also like to acknowledge the support received from the Doctoral Programme in Computer Science and Computer Engineering (DP-CSCE) and the Interdisciplinary Centre for Security, Reliability and Trust (SnT) in multiple occasions.

* "*Life is short, and Art long*", Hippocrates in *Aphorismi*, 400 B.C.E.

Finally, I would like to thank my family for their constant support. My mother Necla and my father Ahmet have always encouraged me to follow the light of science which eventually has brought me to the place where I am now. I am grateful to them so much: “Sevgili anne ve baba, beni bugünlere getirdiğiniz için size çok, çok teşekkür ederim”. My brother İsmail took great responsibilities when I had to leave Turkey for this research. İsmail also achieved so many successes and became a Doctor of Medicine, which gave us another source of happiness: “Sağol paşa”. A very special thanks goes to my dear grandmother Nuriye, who is one of the rare humans who understood the real value of scientific research. This thesis, indeed, is the result of her efforts, starting from my childhood. Thank you, my dear family, for all your support.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

API	Appliction Programming Interface
C&C	Command-and-Control
CBC	Cipher Block Chaining
CNG	Cryptography API: Next Generation
CPU	Central Processing Unit
CRL	Certificate Revocation List
CSPRNG	Cryptographically Secure Pseudo-Random Number Generator
DFG	Data Flow Graph
DGA	Domain Generation Algorithm
DLL	Dynamic Link Library
DRBG	Deterministic Random Bit Generator
Dual EC DRBG	Dual Elliptic Curve Deterministic Random Bit Generator
ECB	Electronic Codebook
F-PRNG	File-based Pseudo-Random Number Generator
FIFO	First-In First-Out
FPE	Format Preserving Encryption
GUI	Graphical User Interface
IAT	Import Address Table
IL	Integrity Level
IoT	Internet of Things
IPC	Inter-Process Communication
IPS	Intrusion Prevention System
IV	Initialization Vector
KDF	Key Derivation Function
KPP	Kernel Patch Protection
KVM	Kernel-based Virtual Machine
MBR	Master Boot Record
MFT	Master File Table
MIC	Mandatory Integrity Control
MS CAPI	Microsoft CryptoAPI
NTFS	New Technology File System
OS	Operating System
OSI	Open Systems Interconnection
OSINT	Open Source Intelligence
PDoS	Permanent Denial of Service
PRNG	Pseudo-Random Number Generator
RNG	Random Number Generator
SSDT	System Service Dispatch Table
SSH	Secure Shell
TLS	Transport Layer Security

UAC	User Account Control
UIPI	User Interface Privilege Isolation
VM	Virtual Machine
VPN	Virtual Private Network
VSS	Volume Shadow Copy Service
XOR	*exlusive-or*

Preliminaries

Introduction 1

Ransomware is a malware, a malicious software that blocks access to victim's data. In contrast to traditional malware, whose break-down is permanent, ransomware's damage is reversible: access to files can be restored after the payment of a ransom, usually a few hundreds US dollars in cryptocurrencies.

Different flavors of ransomware programs exist, but *cryptographic ransomware*, the threat type studied in this thesis, encrypts a victim's files using *strong encryption* [1]. Proper implementation of this strategy gives superior advantage to the cybercriminals, since decryption without the key is infeasible. So the only hope of recovering the files, in the absence of backups, is to pay the ransom. However, paying to the cybercriminals does not guarantee the recovery: according to a recent report, 33.1% of the victims did not get any key in return after the payment, irremediably losing the data and money [2].

While the history of ransomware goes back to earlier, the first strain of modern cryptographic ransomware, `CryptoLocker`, appeared in September 2013. It infected about 500 000 machines and is estimated to collect $3 million from victims [5]. After the proven success of `CryptoLocker`, its business model was adopted by subsequent ransomware variants [6]. Despite being relatively new, this cybercrime has spread fast and become a worldwide pandemic rapidly. According to [7], a US Government's white paper dated June 2016, on average more than 4000 ransomware attacks occurred daily in the USA. This is 300-percent increase from the previous year and such important increment is probably due to the cybercrime's solid business model: with a small investment there is a considerable pecuniary gain which, thanks to the digital currency technology, can be collected reliably and in a way that is not traceable by the authorities.

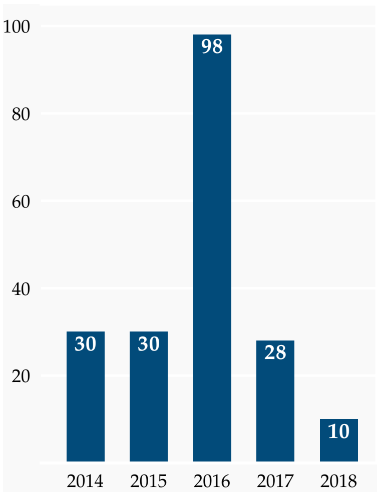
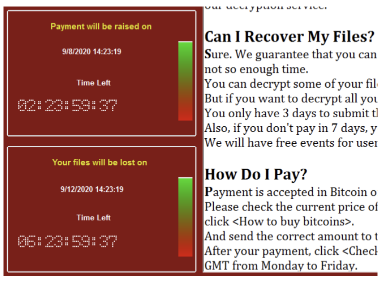


Figure 1.1: The number of new ransomware families in recent years [3, 4].

Notoriety of ransomware peaked in May 2017 when `WannaCry` infected 300 000 computers of, among others, hospitals, manufacturers, banks, and telecommunication companies in about 150 countries. In June 2017, another catastrophic attack was perpetrated by `NotPetya` which caused a global damage that costs approximately $10 billion [8].

Although the volume of infections has dropped after 2017, the attacks got sophisticated and the target of ransomware has moved from individuals to businesses [4]. Unsurprisingly, ransom amounts increased, too. For example, recently, travel management firm CWT paid $4.5 million to hackers after negotiating in an online chat platform [9]. Furthermore, offices of local governments in US became a new target of cybercriminals. Among many others, Atlanta city's computers were compromised by `SamSam` ransomware and the impact was estimated as $17 million [10]. Individuals, businesses, and governments, all combined, cost of global ransomware damage is predicted to reach $20 billion by 2021 [11].



our decryption service.

Can I Recover My Files?
Sure. We guarantee that you can
not so enough time.
You can decrypt some of your fil
But if you want to decrypt all you
You only have 3 days to submit tl
Also, if you don't pay in 7 days, y
We will have free events for user

How Do I Pay?
Payment is accepted in Bitcoin o
Please check the current price of
click <How to buy bitcoins>.
And send the correct amount to t
After your payment, click <Checl
GMT from Monday to Friday.

Figure 1.2: Partial screenshot of the ransom message of `WannaCry`. Note that, to increase the pressure on the victims, notification form includes countdowns to specific payment deadlines.

Evidences demonstrate that encryption is a strong instrument in the hands of criminals. If properly implemented, its impact is irreversible: without knowing the decryption key, recovering the contents of an encrypted file is computationally unfeasible, a very disruptive fact for the victims. However, implementing cryptography flawlessly is a difficult task, and coders of ransomware are challenged by the same issues that have been troubling security engineers in charge of implementing cryptographic applications. One of the most relevant is to generate cryptographically secure encryption keys and keep them safe. Failing in this makes the encryption weak in the sense that it becomes likely to reproduce or retrieve the decryption keys, which would jeopardize the ransomware's business model. In this issue, there is hope as some anti-ransomware solutions (see Section 1.2) indeed offer to recover files counting on cybercriminals' being naïve in implementing ransomware.

Unfortunately, modern ransomware programs are coded more professionally than those in the past. Such professional variants are quite sophisticated, well designed, and properly implemented. Moreover, attacks relying on them are increasing and demanding higher ransom [12]. `NotPetya`, `SamSam`, `Locky`, `Cerber`, `BadRabbit` and `Ryuk` are only a few of the ransomware families that pose serious threats. Bajpai *et al.*, who propose a scale for ransomware similar to the the Saffir-Simpson for hurricanes, classify them as having severity Category 6 [13].

In the absence of an effective cure for the threat, official recommendations suggest prevention. The US Government, for instance, recommends to "regularly back up data and verify its integrity" [14]. Keeping backups however is a solution that does not scale if the threat becomes worldwide: it is an expensive practice that not all companies implement whereas private users are likely not to follow the practice at all. Not surprisingly, a survey on the practice reports that only 42% of ransomware victims could fully restore their data [15].

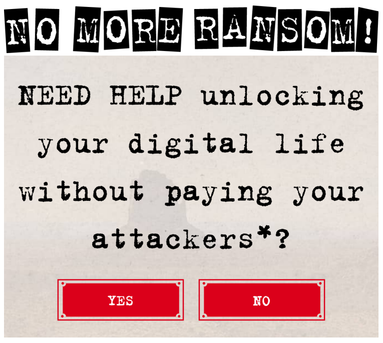

NO MORE RANSOM!
NEED HELP unlocking your digital life without paying your attackers*?
YES
NO

Figure 1.3: The No More Ransom project (`https://www.nomoreransom.org`) aims to help victims recover their data without paying to the cybercriminals. It offers a free service called *Crypto Sheriff* which works as follows. The victim uploads two encrypted files to Crypto Sheriff. Next, the service tries to identify which ransomware variant used in the attack, and queries the database of available decryptors. If there is a match, the decryptor and recovery instructions are shared with the victim. If there is no recovery tools found, the victim is advised to check again later, as new decryptors are added in time.

Security experts have looked into the problem. For example the EUROPOL's European Cybercrime Centre and the Dutch Politie together with Kaspersky Lab and McAfee have founded an initiative called "No More Ransom" whose goal is, we quote, "to disrupt cybercriminal businesses with ransomware connections" and "to help victims of ransomware retrieve their encrypted data without having to pay the criminals" (see Figure 1.3). But, in case of infection, the initiative warns that "there is little you can do unless you have a backup or a security software in place". Other professionals are offering applications that are capable of some protection, but these anti-ransomware systems leverage from existing antivirus/antimalware strategies rather than re-thinking afresh how to solve the problem.

1.1 Anatomy of a Cryptographic Ransomware

Prime target of cryptographic ransomware is the most valuable asset of individuals and companies, the data, which computers usually store in files. Of course, a ransomware attack might include further malicious actions, e.g., spreading over the network and altering Operating System (OS) configuration. In this thesis, we study the cryptographic aspects of ransomware, i.e., methods used for key derivation and encryption.

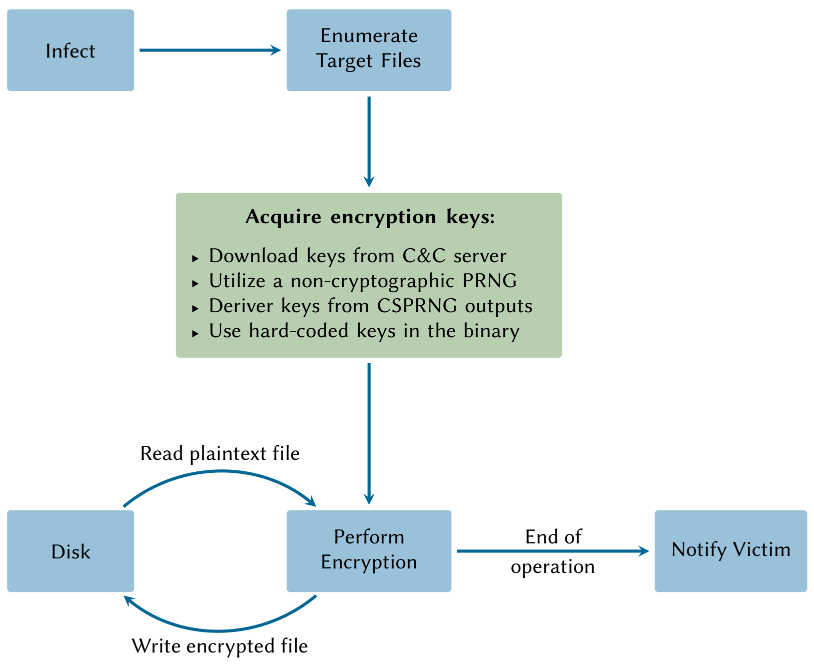


Figure 1.4: Operation flow diagram of a ransomware from a cryptographic point of view.

After infection, a ransomware commences preparation phase, in which it enumerate the files it intends to target and starts acquiring or building the encryption keys. Next, the ransomware proceeds to encrypt files. Once the encryption finishes, it notifies the victim and delivers the ransom demand. Figure 1.4 depicts this operation flow.

To encrypt victim's files, ransomware usually employs a hybrid cryptosystem, that is, an encryption scheme consist of a combination both symmetric and asymmetric algorithms. In order to understand why ransomware authors need to use hybrid cryptosystems, one needs to observe the following facts.

- Encrypting files with solely asymmetric algorithms is a resource intensive task[1]. This might introduce the risk of being detected as high Central Processing Unit (CPU) usage for a long time could trigger anomaly detection systems. Therefore, ransomware must utilize a symmetric algorithm to encrypt files.
- Ransomware needs to use a unique key for each target to prevent victims helping each other [16]. In a mass infection, managing the keys with solely symmetric primitives would not be scalable. Therefore, use of an asymmetric algorithm is required while maintaining the key management of a ransomware campaign.

1: In RSA algorithm, for example, ciphertext c of a message m is computed as

$$c \equiv m^e \pmod N$$

where N is usually selected as a 2048 bit integer (depending on the security level). Modular exponentiation takes significantly more amount of time, especially with large numbers, compared to *exclusive-or* (XOR) or bit shift operations, the main building blocks of symmetric ciphers.

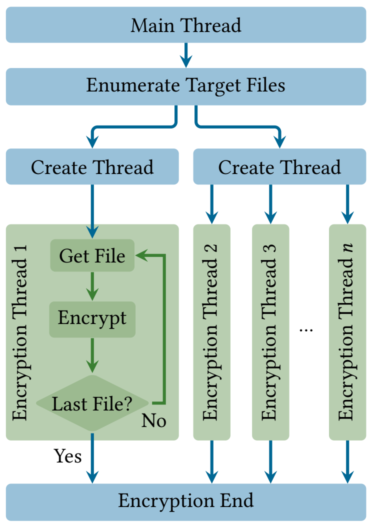


Figure 1.5: To increase the efficiency of the attack, ransomware might utilize multi-threading. One remarkable example is the `Conti` ransomware which creates 32 separate threads to encrypt the files simultaneously [17].

Acquiring Encryption Keys

To achieve long term success, ransomware needs to exercise strong encryption, which requires to use good encryption keys. During the evolution of ransomware, various techniques have been observed to accomplish this task.

One strategy of acquiring the public keys is to fetch them from Command-and-Control (C&C) servers as `CryptoLocker` does [18]. Ransomware can also obtain file encryption keys from these C&C servers, however, usually these keys are derived from the outputs of Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) functions, e.g., `WannaCry` [19]. Another option is to utilize a non-cryptographic Pseudo-Random Number Generator (PRNG) to seed key derivation algorithms. Programming languages in fact provide PRNG functions for developers to utilize in various domains. For example, a ransomware can call `rand` function in C standard library (see Listing 1.2) or use `System.Random` class in C# (see Listing 1.1) language to obtain pseudo-random numbers. This is more frequently observed in samples developed in C# language such as `NegozI` [20] and `Rush/Sanction` families [21]. Lastly, ransomware authors might embed the encryption keys into the malware body before spreading it, e.g., `Cryzip` [22]. These keys can be used directly to encrypt files and thus removing the need for calling any PRNG.

Listing 1.1: The following example illustrates usage of `Random` class in C#, to generate integers between 0 and 100.

```
using System;

var random = new Random();

Console.Write("Outputs: ");
for(int i = 0; i < 5; i++){
   int num = random.Next(100);
   Console.Write("{0} ", num);
}

// [On Console]
// Outputs: 12 8 9 71 1
```

Using the observations above, we identify the following methods that are used by the current generation of cryptographic ransomware to obtain encryption keys:

M1 Derive keys from CSPRNG outputs
M2 Fetch encryption keys from remote servers
M3 Utilize non-cryptographic PRNG to generate secrets
M4 Use secrets embedded into binary executable

Listing 1.2: Using `rand` function in C to generate integers between 0 and 100. Note that we call `srand` to *seed* the RNG with a time-dependent value.

```
#include <stdlib.h>
#include <time.h>

int i, num;
srand(time(NULL));

printf("Outputs: ");
for(i = 0; i < 5; i++) {
    num = rand() % 100;
    printf("%d ", num);
}

// [On Console]
// Outputs: 33 12 9 82 72
```

1.2 Recent Literature in Ransomware Defense

There have been several proposals from the community of information security to mitigate the cryptographic ransomware threat. We can categorize anti-ransomware systems, based on their main strategies, into four groups: *behavioral analysis*, *key escrow*, *decoy files*, and *binary analysis*.

Behavioral Analysis

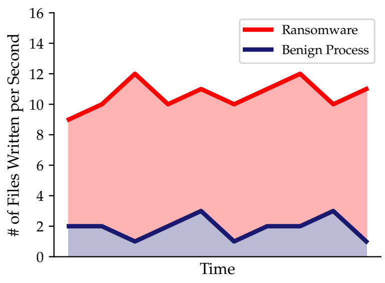
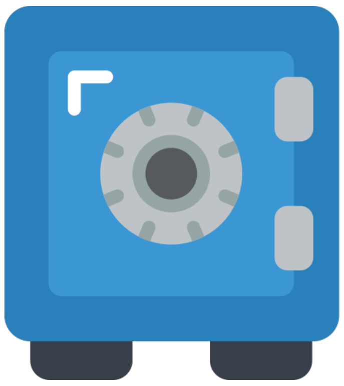


Figure 1.6: *Behavioral analysis* is performed by monitoring a process's activities and interactions with its environments. Typically, ransomware tries to damage as many files as possible in a short period of time. Consequently, statistics like the number of files written per second might significantly increase when ransomware becomes active.

A common anti-malware strategy is to monitor the processes and terminate the ones that exhibit suspicious behavior. The monitored activities include file system access, network connections and interaction with the OS. Among these, the fundamental characteristic of the ransomware is its aggressively encrypting victim's data, causing an unusual file system activity. Using this fact, several defense systems are proposed. One of them, Scaife *et al.*'s CryptoDrop [23] monitors file type changes by looking file headers, compares `sdhash` [24] outputs and measures the Shannon Entropy before and after file writes. Another one, ShieldFS [25] by Continella *et al.* tracks the low-level file system operations and collects the following features: folder listing, file read/write/rename operations, file extension and average entropy of file writes. Comparing these characteristics with that of benign applications allows the detection of ransomware. In addition to detection, ShieldFS creates a copy for each file before a file-write operation, eliminating the potential damage of ransomware. Moreover, Kharraz *et al.* proposed Redemption [26] that also uses the similar metrics for identifying a ransomware activity. However, in contrast to ShieldFS, Redemption redirects file writes to sparse files, rather than creating a full copy of each written file. Differently, *Data Aware Defense* (DaD) by Palisse *et al.* [27] uses chi-square test to determine if the written data is close to random distribution which is indicates that the file is being encrypted. DaD computes the sliding median of this indicator on the last fifty file writes, and suspends the corresponding process that exceeds a predetermined threshold.

Key Escrow

Key-escrow based defense allows the ransomware to complete its attack. This approach is based on the idea that the files encrypted by ransomware can be recovered if the encryption keys can be retrieved after the attack. For this aim, logging the keys used by ransomware is first appeared in the literature by Palisse *et al.* [28], and independently by Lee *et al.* [29]. However, PayBreak of Kolondenker *et al.* [30] was the first to extend the idea to cover the third party cryptographic libraries. In this system, all known cryptographic Appliction Programming Interfaces (APIs) are hooked, cryptographic materials are extracted and securely stored in a key vault. In the case of a ransomware attack, encrypted files are tried to be decrypted by brute-force using the keys and other necessary parameters retrieved from the key vault. A slightly different method, Deterministic Random Bit Generator (DRBG) is proposed by Kim *et al.* to retrieve the random numbers that ransomware used after an attack [31]. DRBG replaces the CSPRNG of the system with a trapdoored PRNG. The trapdoor is known only by the user and is preferably stored in the user's mobile device. After a ransomware incident, this trapdoor is retrieved and given to the PRNG to generate the same outputs that ransomware used. Using these outputs, ransomware's operations are reverted and files are recovered.



Key-escrow is the notion of storing encryption keys in a secure location so that, if needed, the keys can be retrieved at a later time.

Decoy Files

In this strategy, carefully-crafted files are placed as a *decoy* in the file system along with the user's files (see Figure 1.7). These decoys are not supposed to be modified/deleted by the user, so any write request to the decoy files are treated as an indicator of ransomware activity. Monitoring access to decoy files requires considerably less system resource compared to behavioral analysis technique. Lee *et al.* proposed a method to make decoy files for detecting ransomware efficiently [32]. RWGuard [33], a hybrid system proposed by Mehnaz *et al.* also uses decoy files –in addition to behavioral analysis and key-escrow– to mitigate ransomware threat in real time. There are also commercial anti-ransomware programs, e.g., CryptoStopper [34], that uses decoy files to detect ransomware.

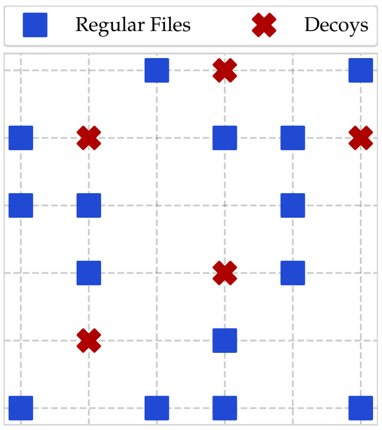


Figure 1.7: *Decoy files* in the file system resemble the land mines placed in the ground. Whenever a process writes to a decoy file, it is immediately considered as a suspicious activity.

Binary Analysis

In this approach, binary programs are analyzed to identify cryptographic operations in their executable codes. To this goal, [35] traces the execution of applications and monitors I/O relationship in the program flow.

Based on the occurrences of *bitwise arithmetic instructions* and *loops*, and relationships between *the inputs and outputs of the program routines*, heuristics are applied to recognize the cryptographic algorithms. On the other hand, [36] uses static analysis and Data Flow Graph (DFG) isomorphisms to identify cryptographic algorithms in the binary programs. Basically, this technique work as follows: First, the DFG of binary program is built. Next, the DFG in hand is normalized using rewrite rules in order to remove the variations due to compiler optimizations. Finally, subgraphs which are isomorphic to graph signatures of cryptographic algorithms are searched in the DFG. A match directly flags that the corresponding algorithm exists in the analyzed program.

Beside technical solutions, Lu and Liao suggest improving user awareness to help mitigate ransomware [37]. Security education for end users would effectively reduce ransomware attacks originating from phishing or spam emails[2]. However, the attack surface that ransomware can exploit is far more larger. As the `WannaCry` attack demonstrates, ransomware evolution has enabled it to spread over the network [38]. Especially, zero-day attacks can amplify the damage of ransomware and user education cannot help in this case.

2: Alerting employees about new threats, and enforcing them to follow security policies and best-practices might lessen the ransomware incidents due to human error.

1.3 Limitations of Current Defense Systems

To begin with, none of the current defenses stop `NotPetya` ransomware. `NotPetya` modifies the Master Boot Record (MBR) of the system and triggers a restart. After the system boots into the malicious kernel installed by `NotPetya`, it encrypts the Master File Table (MFT) of the 'C:\' drive to make data inaccessible while imitating a file system repair, thereby bypasses on-line protections.

Table 1.1: Limitations of the state of the art anti-ransomware systems. *Obfuscation Resilient* denotes that the defense system can detect even obfuscated ransomware programs. *I/O Agnostic* means that the protection does not require to watch the file system activity.

Feature	CryptoDrop	DaD	PayBreak	Redemption	RWGuard	ShieldFS
Mode of Operation	Behavioral	Behavioral	Key-escrow	Behavioral	Hybrid	Behavioral
Obfuscation Resilient	✓	✓	✗	✓	✗	✓
I/O Agnostic	✗	✗	✓	✗	✗	✗
Stops `NotPetya`	✗	✗	✗	✗	✗	✗

Besides, behavioral analysis recognize and stop the ransomware when it is too late. In their experiments over 5100 files, CryptoDrop's authors report that ransomware could encrypt up to 29 files. The median of this statistics reported as 10. The cost of behavioral analysis is another issue. For example, ShieldFS comes with an overhead that has been estimated to exceed 40% while being 26% in average. Similarly, DaD leads to 82% performance loss when writing 4-kilobyte blocks to disk.

Security of benign applications is another issue in the ransomware defense. PayBreak, for instance, must recognize correctly the cryptographic functions employed by the ransomware to log the encryption keys and the parameters. While this is feasible for built-in cryptographic functions on the host system, ransomware that utilizes third-party libraries can bypass detection through *obfuscation*[3]. In addition, there are some issues with the logging of cryptographic APIs. PayBreak logs every key, including

3: *Obfuscation* is the act of transforming a program into an unintelligible program but keeping the program behavior same. We discuss more about obfuscation in Chapter 9.

private keys of Transport Layer Security (TLS) and Secure Shell (SSH) connections. Both protocols offer *forward secrecy*[4] which is build upon employing ephemeral keys. All schemes which count on Layer 6 and above of Open Systems Interconnection (OSI) model (see Figure 1.8) may become vulnerable in this case. PayBreak is designed in such a way that all keys are stored in one place. This may bring the risk of single point of failure, and creates a new target for cybercriminals.

4: *Forward Secrecy* (FS) is the concept of protecting the confidentiality of past communication in the case of exposure of the long term secrets. In key agreement protocols, FS means that even if the long term private keys are compromised, previous sessions keys cannot be obtained by the attacker.

Table 1.1 lists some features of the state of the art ransomware defense systems. In Chapter 3, we review these systems from security perspective.

1.4 Goal and Research Questions

In response to the pandemic of ransomware attacks, security researchers have worked to slow down the threat. Their efforts, however, concentrated into two clusters. The first approach is to adopt the traditional behavioral analysis techniques to use against ransomware, that is, to detect ransomware while it is working. The second approach is to recover the files after they are encrypted by a ransomware, that is, after the attack. Inevitably, both approaches comes with the limitations we stated in Section 1.3. The situation suggest that the combat with ransomware requires a special defense technique which is based on the principles of modern cryptography. This need forms the basis of our research goal:

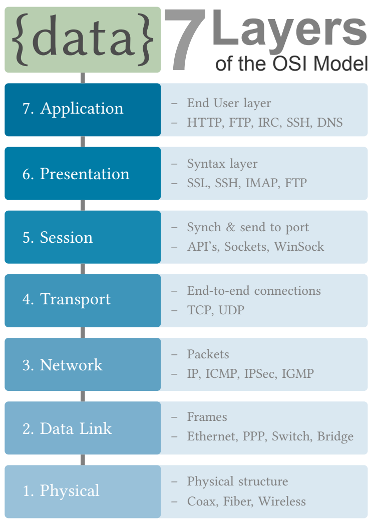


Figure 1.8: OSI model separates communication into 7 layers conceptually. Application layer is the target of most cyber attacks, including those due to ransomware.

Research Goal: *To study the behavior of ransomware, understand its weaknesses, and uncover the cryptographic roots of ransomware to design a defense system which advances the state of the art.*

By approaching the problem from a cryptographic perspective, we can identify the vital requirements for a successful ransomware attack, and take the necessary measures to prevent the damage. The outcome of this research can be regarded as an efficient and effective ransomware defense system that works on end hosts. To achieve this goal, we have structured our research around the following research questions.

RQ1: How can future generations of ransomware may work to bypass current anti-ransomware systems?

The history of malware suggests that new generations of ransomware will be designed to respond to existing protections. We, therefore ask, is there a way to anticipate how the next generation of ransomware will look like? What are the weak points in current defense systems that future ransomware might exploit? How new strains of ransomware might work to bypass these defenses?

RQ2: How secure are the real-world protection mechanisms against novel ransomware attacks?

Although there are several anti-ransomware proposals in the academic literature, most users and companies rely on antivirus (AV) software to protect their digital assets from ransomware attacks. It is therefore essential to analyze and evaluate their security against ransomware. So we ask, can ransomware control whitelisted applications to evade AV programs? Is there any security issue in the AVs so that a ransomware, or in general a malware, can take control of the system?

RQ3: Can ransomware be stopped by controlling secure randomness sources?

To be effective, the ransomware has to implement strong encryption, and strong encryption in turn requires a good source of random numbers. With this insight, we propose a strategy, called UShallNotPass, to mitigate ransomware attacks that considers CSPRNGs as critical resources, controls accesses on their APIs and stops unauthorized applications that call them. Next, we provided an improved version, called NoCry, and tested it against 747 active samples from 56 cryptographic ransomware families to answer the following research questions. Does NoCry stop ransomware before they encrypt any files? Can NoCry protect against zero-day ransomware? What is the overhead cost of NoCry in terms of system resources?

RQ4: What are the implications of emerging technologies on the ransomware phenomenon?

In addition to attacking data by means of cryptography, cybercriminals are likely to find new targets, and design and deploy new strategies, as it commonly happens in security, defenders and attackers will embrace a competition that will never end. In this arm race, anticipating how current ransomware will be affected by the changes in technology may help at least being prepared for some future damage. We therefore ask, what new domains can be appealing for ransomware? Can ransomware infect or damage Internet of Things ecosystems? What are the socio-technical attacks that ransomware might perform? Can ransomware consume online sources for its technological development?

1.5 Thesis Overview

This thesis is structured in ten chapters. Chapter 2 discusses the ethical choices that we had to take in this research and our motivation, and the code of conduct that we commit ourselves to follow. Next, in Chapter 3 through Chapter 9, we explore the topics related to the research questions stated in the previous section. Finally in Chapter 10 we present our closing remarks, describe some future works on the problems that we left open in this thesis. Table 1.2 portrays the chapters and research questions they handle. In the following, we summarize the contents of each chapter and highlight our contributions therein.

Chapter #	**RQ1**	**RQ2**	**RQ3**	**RQ4**
Chapter 3	✓			
Chapter 4	✓	✓		
Chapter 5		✓		
Chapter 6		✓		
Chapter 7			✓	
Chapter 8			✓	
Chapter 9				✓

Table 1.2: Chapters and the corresponding research questions that they help answer.

Chapter 3: Ransomware Evasion Techniques anticipates how future generations of ransomware will work in order to start planning on how to stop them. This chapter contributes to answering **RQ1**. To do so, we study the weak points in the strategies that six of the most advanced anti-ransomware are currently implementing. We identify possible ways in which ransomware might use to bypass these defenses. To prove that our conjecture is in fact more than a thought experiment, we implemented the ransomware samples we have imagined, and prove that it actually passes untouched the anti-ransomware applications that are available to us.

This chapter is based on a conference article [39] which appeared at NordSec 2018.

[39]: Genç, Lenzini, and Ryan (2018), 'Next Generation Cryptographic Ransomware'

Chapter 4: On Deception-Based Ransomware Defense analyzes existing decoy strategies and discusses how they are effective in countering current ransomware by defining a set of metrics to measure their robustness. This chapter contributes to answering **RQ1** and **RQ2**. Here, we adapt an existing threat model of deception to ransomware by defining the measures of quality and confoundedness, which are particularly applicable to ransomware. Theoretical bounds of decoy-based ransomware defenses are also discussed in this chapter.

This chapter is based on a conference article [40] which appeared at DIMVA 2019.

[40]: Genç, Lenzini, and Sgandurra (2019), 'On Deception-Based Protection Against Cryptographic Ransomware'

Chapter 5: Vulnerability Analysis of Real-World Systems demonstrates two novel attacks that malware can use against AVs. This chapter contributes to answering **RQ2**. Our first attack consists in simulating mouse events to control AVs, namely to send them mouse "clicks" to deactivate their protection. We prove that many AVs can be disabled in this way. The second attack consists in controlling white-listed applications, such as `Notepad`, by sending them keyboard events (such as "copy-and-paste") to perform malicious operations on behalf of the malware. We prove that the anti-ransomware protection feature of AVs can be bypassed if we use `Notepad` as a "puppet" to rewrite the content of protected files as a ransomware would do.

This chapter is based on two academic contributions: a conference article [41] which appeared in ACSAC'19 and a manuscript [42] which extends it. The manuscript has been submitted to the journal *Digital Threats: Research and Practice* and is in press.

[41]: Genç, Lenzini, and Sgandurra (2019), 'A Game of "Cut and Mouse": Bypassing Antivirus by Simulating User Inputs'

[42]: Genç, Lenzini, and Sgandurra (), 'Cut-and-Mouse and Ghost Control: Exploiting Antivirus Software with Synthesized Inputs'

Chapter 6: Sandbox Evasion describes a technique - unprecedented, we discovered it while analyzing a ransomware sample, used to evade sandbox detection. This chapter contributes to answering **RQ2**. Analyzed in a Cuckoo Sandbox, the sample was able to avoid triggering malware indicators, thus scoring significantly below the minimum severity level. Here, we discuss what strategy the sample follows to evade the analysis, proposing practical defense methods to nullify, in our turn, the sample's furtive strategy.

The contents of this chapter is based on a conference article [43].

[43]: Genç, Lenzini, and Sgandurra (2019), 'Case Study: Analysis and Mitigation of a Novel Sandbox-Evasion Technique'

Chapter 7: Stopping Ransomware by Controlling CSPRNGs provides the design and evaluation of our ransomware defense system, UShallNotPass, which works by establishing an access control over CSPRNG of the OS. This chapter contributes to answering **RQ3**. Here, we show that acquiring encryption keys securely is a vital task for ransomware. We prove that by blocking access to good randomness sources, ransomware can be stopped efficiently and effectively. Experimental results which include benchmarks and some usability tests are also provided in this chapter.

This chapter is based on two conference articles: [44] and [45].

[44]: Genç, Lenzini, and Ryan (2018), 'No Random, No Ransom: A Key to Stop Cryptographic Ransomware'

[45]: Genç, Lenzini, and Ryan (2018), 'Security Analysis of Key Acquiring Strategies Used by Cryptographic Ransomware'

Chapter 8: Efficient End-Host Protection Against Ransomware presents NoCry, an improved design and implementation of our anti-ransomware idea. This chapter contributes to answering **RQ3**. We discuss our new solution that is more secure (with components that are not vulnerable to known attacks), more effective (with less false negatives in the class of ransomware addressed) and more efficient (with minimal false positive rate and negligible overhead) than the original, bringing its security and technological readiness to a higher level.

The contents of this chapter is based on a workshop article [46].

[46]: Genç, Lenzini, and Ryan (2020), 'NoCry: No More Secure Encryption Keys for Cryptographic Ransomware'

Chapter 9: Possible Future of Ransomware points out new possible ransomware targets and attack types inspired by the cybersecurity incidents occurred in real-world applications. This chapter contributes to answering **RQ4**. In this respect, we described possible threats that ransomware may pose by relying on novel techniques, like rootkit access, obfuscation, and white-box cryptography, not yet adopted in real attack as well as by targeting critical domains, such as the Internet of Things and the Socio-Technical systems, which will worrisomely amplify the effectiveness of ransomware attacks. Finally, we provide evidence that ransomware authors utilize open source software for their nefarious goals.

This chapter is based on two conference articles: [47] and [48].

[47]: Genç, Lenzini, and Ryan (2017), 'The Cipher, the Random and the Ransom: A Survey on Current and Future Ransomware'

[48]: Genç. and Lenzini. (2020), 'Dual-use Research in Ransomware Attacks: A Discussion on Ransomware Defence Intelligence'

Chapter 10: Closing Remarks and Future Work presents considerations taken in conclusion to this research project. We revisit our goal and questions, and discuss how this thesis addresses them. We also present some future works and open problems which remain to be explored.

1.6 Scientific Contributions

Below is the list of scientific papers arose from this thesis.

Publications on ransomware as main author

1. Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. 'The Cipher, the Random and the Ransom: A Survey on Current and Future Ransomware'. In: *Proceedings of the Central European Cybersecurity Conference 2017*. CECC 2017. Ljubljana, Slovenia: University of Maribor Press, 2017
2. Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. 'No Random, No Ransom: A Key to Stop Cryptographic Ransomware'. In: *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment*. DIMVA 2018. Cham: Springer, 2018
3. Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. 'Next Generation Cryptographic Ransomware'. In: *Proceedings of the Secure IT Systems - 23rd Nordic Conference*. NordSec 2018. Cham: Springer, 2018
4. Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. 'Security Analysis of Key Acquiring Strategies Used by Cryptographic Ransomware'. In: *Proceedings of the Central European Cybersecurity Conference 2018*. CECC 2018. New York: ACM, 2018
5. Ziya Alper Genç, Gabriele Lenzini, and Daniele Sgandurra. 'On Deception-Based Protection Against Cryptographic Ransomware'. In: *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment*. DIMVA 2019. Cham: Springer, 2019
6. Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. 'NoCry: No More Secure Encryption Keys for Cryptographic Ransomware'. In: *Proceedings of the Emerging Technologies for Authorization and Authentication*. Cham: Springer, 2020
7. Ziya Alper Genç, Gabriele Lenzini, and Daniele Sgandurra. 'Case Study: Analysis and Mitigation of a Novel Sandbox-Evasion Technique'. In: *Proceedings of the Third Central European Cybersecurity Conference*. CECC 2019. New York: ACM, 2019
8. Ziya Alper Genç, Gabriele Lenzini, and Daniele Sgandurra. 'A Game of "Cut and Mouse": Bypassing Antivirus by Simulating User Inputs'. In: *Proceedings of the 35th Annual Computer Security Applications Conference*. ACSAC '19. New York: ACM, 2019
9. Ziya Alper Genç. and Gabriele Lenzini. 'Dual-use Research in Ransomware Attacks: A Discussion on Ransomware Defence Intelligence'. In: *Proceedings of the 6th International Conference on Information Systems Security and Privacy*. ICISSP 2020. Setúbal, Portugal: SciTePress, 2020
10. Ziya Alper Genç, Gabriele Lenzini, and Daniele Sgandurra. 'Cut-and-Mouse and Ghost Control: Exploiting Antivirus Software with Synthesized Inputs'. In: *Digital Threats: Research and Practice* (). In press

Other contributions made in parallel to this thesis

11. Ziya Alper Genç et al. 'Examination of a New Defense Mechanism: Honeywords'. In: *Information Security Theory and Practice*. Cham: Springer, 2018
12. Ziya Alper Genç et al. 'A Security Analysis, and a Fix, of a Code-Corrupted Honeywords System'. In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. ICISSP 2018. Setúbal, Portugal: SciTePress, 2018

13. Ziya Alper Genç et al. ‘A Critical Security Analysis of the Password-Based Authentication Honeywords System Under Code-Corruption Attack’. In: *Information Systems Security and Privacy*. Cham: Springer, 2019
14. Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. ‘Method for Preventing Ransomware Attacks on Computing Systems’. Pat. WO2020002203A1. Université du Luxembourg. Jan. 2, 2020
15. Ziya Alper Genç, Vincenzo Iovino, and Alfredo Rial. ‘“The Simplest Protocol for Oblivious Transfer” Revisited’. In: *Information Processing Letters* 161 (2020)

Ethical Considerations | 2

Working on the ransomware threat by pointing out the potential limitations in current anti-ransomware defences raises an ethical question: could these insights be misused?

This ethical issue can be related to dual-use of research, which is mentioned in Article 2 of Council Regulation (EC) No 428/2009 [49] and its amendment Council Regulation (EU) No 388/2012. It defines dual-use items as "items, including software and technology, which can be used for both civil and military purposes [...]". Recently the EC has released a guidance note [50], where it comments on "Misuse of research", which has to be understood as "research that can generate knowledge, materials, methods or technologies that could also be used for unethical purposes" and in this phrasing, we recognize the ethical matter of our research. In adherence to the guidance, we comment on the risks of our research and we state ourselves that we behave to reduce the risk of misuse. By pointing out the potential and theoretical weaknesses in current anti-ransomware strategies, we may give suggestions on how to improve current variants, but we also warn cyber-security analysts and help them proactively to improve current anti-ransomware. It must be said that we work not by discovering bugs in applications—disclosing them will immediately have negative consequences. We rather discuss what we think are limitations in specific approaches against ransomware. Thus, using our arguments to build a fully-fledged malware requires to fill a non trivial knowledge and technological gap.

Whenever, in support to our research, we implement some piece of software to test a specific anti-ransomware application, we do not disclose any code. This removes the risk that it may be re-used inappropriately. At the same time, we dutifully inform of our findings the authors of the application that we have put to a test. We invite them to challenge our arguments and evidence and we warn them that, were our speculations true, there could be a way to circumvent what they propose as a defence. We hope in this way to contribute to improve it too.

2.1 Coordinated and Responsible Disclosure

This research has ethical concerns of dual use research: our findings can be used to improve the security model of AVs as well as to attack them. Aware of the risk, we adhere to an ethical code of conduct [51]: we do not disclose the names of the AV companies, nor publicly share any piece of software that can be used to exploit the vulnerabilities reported in this thesis. We also follow a practice of responsible disclosure [52]: we have dutifully engaged with the affected AV companies to inform them about our findings, ensuring they know about our research informing. We are sharing with them whatever they need to replicate our attacks and to gain insights to fix the vulnerabilities that we think are the root cause of the success of the attacks. All the affected AV companies have been dutifully informed, although some of them haven't replied to us yet: we do hope we will be acknowledged by all AV companies in due time. We told them that, we believe, is their the responsibility to fix whatever needs to be fixed, but we leave to them to judge the severity and the impact of our research on their products and clients. We do not force or nudge them to react by using our research as a leverage. Such a mission is outside to our ethical stand-point. In any case, we will not reveal the names of AVs as with some of them.

That said, we think useful to report some data about how many AVs we have found vulnerable, and how they have responded to our attempts, to contact them. Table 2.1 summarizes the situation for the companies that have, at the time of writing, September 2020, answered us. So far, we identified that AV products of 14 vendors are vulnerable to our attacks. Ten (10) out of fourteen (14) vendors have engaged in a conversation with us, or have answered somehow.

Please note that we have only engaged with the vendors of the 14 AVs we discovered vulnerable. Therefore, we do not know whether these attacks are successful (or not) on other AVs, but we do hope that the results of our research motivate other AV vendors to perform a similar security analysis and, in case, adopt countermeasures.

Table 2.1: The process and results of the responsible disclosure with the affected AV vendors. Vx denotes the vendor of AVx.

Vendor	Dedicated Channel	Encrypted Email	Disclosure Platform	Disclosure Date	Response Time	Current Decision
V4	✓	✓	✗	30.10.2019	1 day	Released a fix
V5	✗	✗	✗	09.09.2020	Still Waiting	
V6	✓	✗	✗	09.09.2020	< 1 day	Working on
V7	✓	✗	✗	30.10.2019	1 day	Rebutted
V8	✓	✗	✗	09.09.2020	3 days	Won't Fix
V12	✗	✗	✗	09.09.2020	< 1 day	
V14	✗	✗	✗	09.09.2020	Still Waiting	
V16	✓	✗	✓	30.10.2019	1 day	Not Prioritized
V20	✓	✓	✗	09.09.2020	Still Waiting	
V24	✓	✓	✓	10.09.2020	4 days	
V26	✗	✗	✗	09.09.2020	Still Waiting	
V27	✓	✓	✗	30.10.2019	5 days	Fixed
V28	✗	✗	✗	30.10.2019	No Response	
V29	✗	✗	✗	30.10.2019	1 day	Working On

Communication Method We strove to communicate over official channels to inform the vendors, and share the technical details of the issues, proof-of-concept materials, as well as potential mitigation techniques. As we can see from Table 2.1, V16 and V24 use independent platforms for vulnerability disclosure. We filled application forms and marked the vulnerability type as a *critical*. Four vendors —V4, V20, V24, and V27— accept encrypted emails and their PGP keys are available on the company websites. We used this communication method and sent encrypted emails to these addresses (except V24, which was reached using an independent platform). V6 and V8 accepts unencrypted emails (V6 publishes a PGP key, but it does not match the email address dedicated for vulnerability disclosure). V7 provides a dedicated web page for vulnerability reports, so we filled the form on the page with a description of the issue. V5, V12, and V26 do not provide a dedicated channel to report vulnerability, therefore, we engaged with these companies using their support mail. Lastly, V14, V28, and V29 do not have any suitable channel to reach out, thus we filled the contact forms at the company websites as a last resort.

Reactions Six out of fourteen vendors replied to our disclosure in 1 day (hereafter day means calendar day). However, no vendor could give us an exact time for a fixed release. V4 is the only vendor that informed us that it released the fixed version. V27 worked with us to fix the vulnerability and implemented the patch, however, did not inform us if it has released the fixed version. Quite surprisingly, vendor of AV16 acknowledged that V16 is vulnerable but they considered its severity as low, and did not put a patch for it their priority list. V8 also informed us that the issue will not be addressed by V8 as it should be fixed in the OS level. V6 and V29 replied that they acknowledged the vulnerability and are working on the issue, however, did not inform us about any release plan. V7 rebutted our initial disclosure; we replied back and wait for their response. We are in active communication with V12 and V24 about the issue. We attempted to contact V28 several times for 316 days but it never replied back to us. In our last attempt, we created a ticket in their support system and described the vulnerability. V28 closed the ticket but did not reply. We have not received any reply from V5, V14, V20, and V26.

V4 an V27 offered to publish our names on the company websites. Furthermore, V4 offered us a bounty for our vulnerability disclosure. Microsoft has been informed about our *Cut-and-Mouse* attack since the beginning. They acknowledged the weakness in Windows 10 and informed us that an investigation is going on to find the root cause of the weakness and to formulate an efficient fix.[1]

1: Note that, in this specific case, it is not possible to anonymize the name of the affected vendor as the root of the vulnerability that enables one of our attacks lies in the Operating System running the AVs (Windows) rather than the AVs. At the time of the writing, not only the responsible disclosure period, 90-days, has elapsed but Microsoft is working towards a fix for this, so we feel compelled to share the details publicly.

2.2 Ethical Code of Conduct

We are aware that the research we have ourselves embarked may give ideas to criminals. But there is no reason to believe that criminals will not have those ideas by themselves. In the history of malware (see e.g., [53]) criminals have always tried to be one step ahead; besides, our research has nothing fancy and it does not contain such an inventive step that cannot be reproduced by others. It more humbly roots into how cryptography works. However, even with this premise, we questioned ourselves about how to do this research ethically.

As we anticipated in the introduction, working with malware raises ethical questions [54], although we have not involved people in our research, nor we have collected personal or sensitive data or attacked real operating systems, nor were we involved in any conversations with criminal associations or victims, actions which would have required us following specific guidelines as discussed in [55].

Despite having conducted our research in isolation, we agree with Rogaway's "The Moral Character of Cryptographic Work" [56] when he suggest to "be introspective about why you are working on the problems you are". We hope to have motivated sufficiently why we started this research pathway in the first place. At the same time we informed ourselves about the University of Luxembourg Policy on Ethics in Research[2]; it suggests that researching on protection against computer viruses is at risk of dual use. The guidelines recommend researchers to "report their findings responsibly", but there is no indication that may suggest what is a responsible behavior. As well there are no guidelines in the ACM Code of Ethics and Professional Conduct[3], another manifesto we looked into. It suggests principles, like "Avoid harm" and "Ensure that the public good is the central concern during all professional computing work" but how to comply with those principles is not told. The EU "Regulation No 428/2009" considers software as a dual use item, so we are certain that there are ethical consideration to address. Most of the literature on dual-use refers to life science and cannot be migrated to computer science but the EU's "Ethics for researchers" [57] suggests something general that can be useful in our case: "special measures need to be taken to ensure that the potential for misuse is adequately addressed and managed". Thus we decided to set up our own ethical practise which consist in embrace two important measures: (i) *Responsible Disclosure*: before submitting camera ready version, we informed all parties affected by the vulnerabilities that we think we have disclosed in this thesis, giving them all details about the flaws and the potential attacks. We hope in this way to warn awareness in the scientific community, and in particular in the researchers that engineered the defences whole limitations we have discussed; (ii) *Safe Handling of Hazardous Code*: we determined ourselves not to share any portion of the source code with the public, not to send it unsecured in using insecure channels (e.g., emails) and to keep it stored in an encrypted disk. At the same time all experiments have been done with a machine whose access is strictly limited to the researchers involved.

2: For more information, please visit https://wwwen.uni.lu/research/chercheurs_recherche/standards_policies

3: Available at https://www.acm.org/code-of-ethics

Security Analysis of Anti-Ransomware Systems

3 Ransomware Evasion Techniques

Cryptographic ransomware families share a common goal: to encrypt a victim's files. They also share a few fundamental tasks that they necessarily have to execute to achieve this goal. For instance, they have to manage encryption and decryption keys; and they have to read, encrypt (and if the victim is lucky) decrypt, and write files. However, cryptographic ransomware comes in different forms. Although constrained to perform those common steps, they can reach the goal in different ways, so giving raise to different families of them.

For the same reason there are also many potential, not all necessarily effective, strategies to counteract ransomware. Current anti-ransomware approaches implement mainly two strategies: *key-oriented protection* and *behavioral analysis.*

3.1 Anti-Ransomware Approaches

Key-oriented Protection (KP). The rationale of those who follow a key-oriented protection strategy is that ransomware needs encryption keys and therefore it is better to keep those keys under control. "Keep keys under control" is not a simple action; current solutions have interpreted and implemented it in at least three distinguished methods:

(**KP**-i) - *placing backdoor in CSPRNG.* In this strategy, a trapdoor is inserted to the CSPRNG of the host system. The aim of this trapdoor is to enable reproducing the previous outputs of CSPRNG for a given time. Thus, the random numbers used by ransomware as a seed can be obtained after an attack. Using these seed values, the keys used by ransomware are re-derived and the files are restored. In [31], Kim *et al.* proposed this technique to mitigate ransomware.

(**KP**-ii) - *escrowing encryption keys.* In this approach, cryptographic API are hooked, encryption keys and other parameters are acquired, and stored in a secure location. After a ransomware incident, these materials are used to recover the files. The first key-escrow based ransomware defense systems are proposed independently by Lee *et al.* [29] and Palisse *et al.* [28] and focused on only the built-in cryptographic API. Later, PayBreak [30] extended this technique to include the functions in third-party cryptographic libraries.

Behavioural Analysis (BA). Defenses that implement behavior analysis, monitor the interactions of applications and measure certain factors that may indicate the presence of a ransomware activity. Solutions diversify depending on the indicators used to monitor for the presence of ransomware. We recognize four major methods:

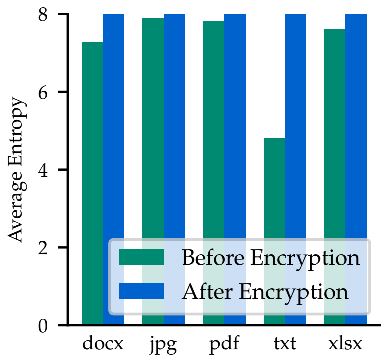


Figure 3.1: Average entropy of different file types, before and after being encrypted. The values are computed using a subset of author's files.

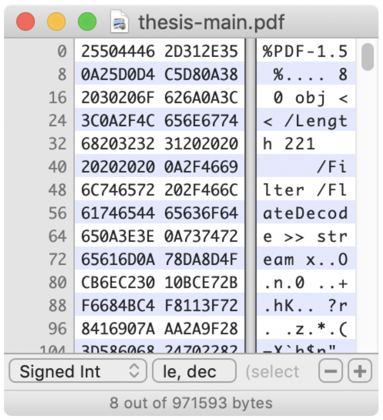
(**BA**-i) - *measuring entropy inflation.* Encryption increases the entropy of the files (see Figure 3.1). Therefore, encryption can be detected by measuring the entropy of files, before and after file-write operations. A rough estimate of the entropy e, of a byte array $(x_i)_{i=1}^{n}$ that is often used is Equation 3.1.

$$e = \sum_{k=0}^{255} P_k^x \log_2 \frac{1}{P_k^x} \quad \text{where} \quad P_k^x = \frac{|\{i : x_i = k\}|}{n} \tag{3.1}$$

Monitoring entropy changes is a method commonly used by CryptoDrop [23], ShieldFS [25], and Redemption [26].

(**BA**-ii) - *detecting content modification.* Modern cryptographic algorithms produce ciphertext that completely differs from the plaintext data. Therefore, if the similarity between original file and modified file is small, the file might have been encrypted. In this respect, CryptoDrop utilizes `sdhash` [24] tool to compute dissimilarity of files to detect encryption performed by ransomware.

(**BA**-iii) - *identifying file-type changes.* File type can be identified by position-sensitive tests, e.g., reading byte values at specific locations in a file. In contrary to benign applications, ransomware changes this information when encrypting a file, transforming the file into an unknown type. Therefore, changing file types is a strong indicator of ransomware activity. For example, CryptoDrop uses the same technique of `file` [58] utility to detect change of file types.



```
  0  25504446 2D312E35  %PDF-1.5
  8  0A25D0D4 C5D80A38   %.... 8
 16  2030206F 626A0A3C   0 obj <
 24  3C0A2F4C 656E6774  < /Lengt
 32  68203232 31202020  h 221
 40  20202020 0A2F4669        /Fi
 48  6C746572 202F466C  lter /Fl
 56  61746544 65636F64  ateDecod
 64  650A3E3E 0A737472  e >> str
 72  65616D0A 78DA8D4F  eam x..O
 80  CB6EC230 10BCE72B  .n.0 ..+
 88  F6684BC4 F8113F72  .hK.. ?r
 96  8416907A AA2A9F28  .  .z.*.(
```

Figure 3.2: Hexadecimal encoding of leading bytes of a PDF file (in fact, a draft of this thesis). When interpreted as text, the first 5 bytes of PDF files appears as `%PDF-`, of which, the hexadecimal equivalent is `0x25 0x50 0x44 0x46 0x2d`. This array of hex values is called *Magic Number* (or *Magic Bytes*) and can be used to identify a file type.

(**BA**-iv) - *testing goodness-of-fit.* Encryption produces data which have a pseudo-random distribution. Based on this fact, DaD [27] employs chi-squared goodness-of-fit test to determine if the written data is close to random distribution and conclude that the file is being encrypted. To this aim, observed byte array is put into a frequency histogram with class interval 1 from 0 to 255. Let N_i denote the number of variates in bin i, and n_i be a known distribution. The chi-squared test value of this array is computed as in Equation 3.2

$$\chi^2 = \sum_i \frac{(N_i - n_i)^2}{n_i} \tag{3.2}$$

Indicators do recognize ransomware activities but also benign applications. For example, file compression utilities might show similar patterns with ransomware. False positives can be reduced by combining multiple indicators, e.g., CryptoDrop utilizes (**BA**-i), (**BA**-ii) and (**BA**-iii).

In addition to the indicators above, there are other behavioral features such as file access patterns e.g., read/write/delete operations, and access frequency that can be used to identify ransomware activity. ShieldFS [25] and RWGuard [33] employ these type of indicators in their custom detection logic. While these systems claim to have highly accurate detection rates, in [59] authors examine them, and show that ransomware can evade detection by distributing the overall workload of malicious operations across a small number of processes (see Figure 3.3). Since these systems are already found vulnerable, we did not include them in our security analysis.

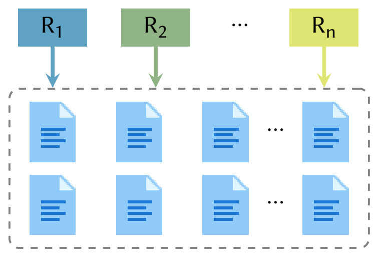


Figure 3.3: Process & function splitting strategy. Each ransomware process carries out a small portion of the attack so that they all stay under the radar while the files are being encrypted.

The analyzed systems in this chapter and their corresponding defense techniques are given in Table 3.1.

System	(**KP**-i)	(**KP**-ii)	(**BA**-i)	(**BA**-ii)	(**BA**-iii)	(**BA**-iv)
Kim *et al.* [31]	✓					
CryptoDrop [23]			✓	✓	✓	
Lee *et al.* [29]		✓				
Palisse *et al.* [28]		✓				
DaD [27]						✓
PayBreak [30]		✓				

Table 3.1: Select anti-ransomware systems and their main defense methods.

3.2 Vulnerability Analysis of Countermeasures

"Every law has a loophole" says an old proverb, meaning that once a rule is known, it becomes known also how to evade it. This holds true also in the ransomware *versus* anti-ransomware arms race and in both ways. Knowing how ransomware works, one can design more effective defenses; knowing how defenses work, one can design more penetrating ransomware. In this section we discuss potential limitations in current anti-ransomware, and we imagine and discuss how future generation ransomware could evolve to overcome those defenses.[1]

1: In this exercise, we apparently take the side of ransomware but the goal is to stimulate the scientific community to anticipate better defenses that can work not only against current ransomware but also against forthcoming generation of them. This choice is not exempt from consequences. We discuss in Chapter 2 the ethical aspects in this research and we comment on the code of conduct we have committed ourselves to in developing this work.

Limits of Key-Oriented Protection

Key-oriented protection defenses aim at to prevent ransomware from using, undisturbed, cryptographic API.

In this respect, (**KP**-i) inserts a backdoor into CSPRNG API. A ransomware may evade this defense by using an alternative source of randomness. The critical question is whether there exist sources of randomness that are as good as CSPRNG. We will elaborate more on this approach in Section 3.3.

Instead, (**KP**-ii) logs parameters and outputs of CSPRNG, built-in cryptographic API, and *recognized* functions in third-party libraries. As stated in [30], the critical limitation of this approach is that recognizing statically linked functions from third-party libraries is sensitive to *obfuscation*. Obfuscation does not affect recognizing calls to built-in API, so evasion is possible when ransomware binary is obfuscated and the ransomware refrain from using built-in API.

Limits of Behavioural Analysis

To detect cryptographic activities, behavioral analysis uses indicators, which are features revealing the presence of certain suspect behaviours; it also relies on constantly applying measurements and tests on files, before and after I/O operations.

In this respect, (**BA**-i) tests if the entropy of the file increases during a write operation using Equation 3.1. It assumes that the encryption always increases the Shannon Entropy of a file. Indeed, this assumption holds for standard ciphers such as AES [60]. The entropy inflation test can be bypassed by changing the encryption algorithm with a one that preserves the entropy of the blocks.

Likewise, (**BA**-ii) compares the contents of a file before and after a file write operation and checks if the similarity score is above a threshold. A fully encrypted file should look like a random data and the comparison should yield a score close to 0, indicating a strong dissimilarity. This is true if the whole file is encrypted. A partially encrypted file, when compared with the plaintext version, is likely to result in high similarity scores: (**BA**-ii) may not be triggered while the file becomes practically unusable.

(**BA**-iii) can also be easily bypassed. If ransomware saves the file header, i.e., does not encrypt the lead bytes of the file, and encrypts the rest, than the output of probe for file-type remains same. It should be noted that this information is generic, i.e., publicly available, therefore cannot be considered as a critical data. Consequently, ransomware would not lose any profit by omitting the file-type identifying bytes. To nullify this strategy, anti-ransomware systems may utilize context-sensitive tests which scan entire file to detect a file's type, with the expense of degraded performance. In the experiments (Section 3.5), however, we haven't encountered such a detection. We remark that this defense might be bypassed by adding read-/write routines for specific target file types, which is an implementation effort.

Finally, (**BA**-iv) tests if the written data is close to random distribution, based on the observation that standard ciphers like AES produce randomly distributed outputs. For this aim,chi-squared test given in Equation 3.2 is used. However, if the chi-squared values can be kept constant during the obfuscation of file, this indicator will not trigger the alarm.

3.3 Future Ransomware Strategies

We present the blueprints of two novel ransomware samples that we claim are able to evade the defense systems listed in Table 3.1. The architecture of the samples is similar to that of `WannaCry` from the point of key management. That is, each file in the victim's computer is encrypted with a unique symmetric key. Moreover, these symmetric keys are encrypted with a public key generated on the victim's computer. The corresponding secret key is then encrypted with the master public key embedded in the binary executable.

While this approach brings the risk of private key's being captured, it also removes the necessity of active connection to our hypothetical C&C server which might be blocked by network firewalls and cause ransomware to fail[2].

2: Network level defense works best against hardcoded addresses and efficiently detects the usage of Domain Generation Algorithm (DGA). To remove this burden, `Cerber` ransomware utilizes transaction information in the bitcoin network to locate its C&C server [61].

Bypassing Key-Oriented Defenses

Our first construction targets key-oriented defense systems. As we point in Section 1, (**KP**-i) can be bypassed by utilizing an alternate randomness source. However, to defeat (**KP**-ii) completely, it is also required to statically link against a third-party library and apply obfuscation.

Deriving Encryption Keys

A simple technique to generate the file encryption keys that malware might adopt is what is known in cloud computing circles as *Convergent Encryption* [62]. Here, the cryptographic keys are derived from files themselves. A simple implementation is as follows. Let E be an encryption algorithm, H be a hash function, and F be the file. The technique consists in deriving the encryption key by hashing the file itself, that is $H(F)$. The resulting encryption is therefore $E(F, H(F))$.

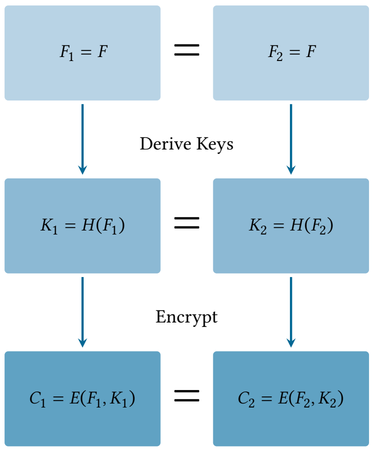


Figure 3.4: Given the same plaintext, convergent encryption schemes produce the same ciphertext.

The technique is free from the issues that may arise in the cloud computing. While convergent encryption is useful in certain scenarios, in the context of cloud computing, this technique may leak information as follows. For publicly-available plaintext files, the adversary can check and learn if the ciphertext belongs to these files. However, this is not really an issue in the context of ransomware: if the user still has the plaintext file(s), say in a backup, then the ransomware will not be effective anyway.

Our hypothetical ransomware thus computes $H(F)$ and derives the key by truncating this hash value to the length of K. This allows to evade (**KP**-i). To win against (**KP**-ii), we need a little more care: H and E must be statically-linked against a third-party library and obfuscated, otherwise (**KP**-ii) can acquire and store the result of H where K lies therein. The same requirement also applies to E. However, having a hash function in hand, the necessity of a block-cipher can also be fulfilled in the context of ransomware.[3]

3: As we mentioned in Chapter 1, modern ransomware employs battle-tested encryption algorithms to make it infeasible for victim to decrypt the files without paying for the keys. However, the prime goal of a ransomware attack is not to make decryption impossible without the keys. Rather, it is to increase the cost of recovery so that paying the ransom appears to be a better option. Actually, an encryption algorithm that is strong enough to make recovery efforts cost more than the ransom amount might be sufficient for the ransomware business.

Symmetric Key Encryption Method

Once the ransomware has got hold of good grade encryption keys then it can employ various well-established symmetric encryption techniques to the victim's files, for example a stream cipher, e.g., based on a hash function in counter mode, or block cipher in an appropriate mode, e.g., chained. The exact choice of algorithm is not so important as long as it sufficiently cryptographically strong to render cryptanalysis significantly more expensive than paying the ransom. However the algorithms should be fairly simple so as to be coded compactly and easy to obfuscate.

To encrypt the files we built a stream cipher using a keyed hash function build from H. Our construction utilizes H to generate a keystream in a similar way to the counter-mode of block ciphers. The keystream and the plaintext are combined using the XOR operation.

Let F be a plaintext stream such that $F = P_1 \,\|\, P_2 \,\|\, \dots \,\|\, P_n$ where each P_i has equal bit length to output of H, except possibly P_n, and $K = H(F)$. Encryption of F is done as follows:

$$S_i = H(K \,\|\, i)$$
$$C_i = P_i \oplus H(K \,\|\, S_i)$$

for $i = 1, 2, \dots, n-1$. For $i = n$, $H(K \,\|\, S_n)$ is truncated to the length of P_n.

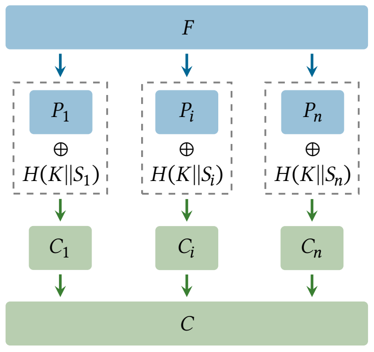


Figure 3.5: Using a keyed hash function in counter mode for encryption.

In our design, we assume that H is (i) one-way: given K, it should be hard to find F such that $H(F) = K$; and (ii) collision-free: it should be hard to find $S_i \neq S_j$ such that $H(K \,\|\, S_i) = H(K \,\|\, S_j)$ (iii) pseudo-random: it is difficult to guess $H(K \| i)$ —in our implementation, i has a fixed length of 32 bits— without knowing $K \| i$.

Voiding Memory Dump Analysis

Current software implementations of symmetric cryptographic algorithms require the encryption keys to be retrieved during the execution. Consequently, when encrypting files, the encryption keys reside in the memory area of the ransomware[4] process. Using this observation, defense techniques emerged (e.g., [63]) which try to dump the memory of the encrypting process and extract the keys to roll-back the damage.

4: Actually, ransomware might try to inject malicious code into other processes. In this case, memory of the encrypting process is dumped.

Deriving keys from the files' hashes overcomes this defense, as different files will result in distinct encryption keys. If a defense system detects files being encrypted, suspends the process and extracts the keys, it can only decrypt the file which is currently being accessed. Previous files cannot be recovered anymore as they are encrypted with different keys which were already destroyed at the time of detection.

File based PRNG

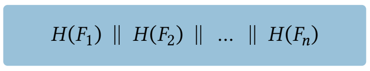


Figure 3.6: Initial state of our F-PRNG. The pool is filled by the concatenating hashes of a subset of the target file list.

We have developed a PRNG which inputs files, outputs pseudo-random bytes and provides the sufficient functionality for the purposes of ransomware. The PRNG has a pool, which is implemented as a byte array and initially filled with the hashes of files that will be encrypted. As the ransomware needs n bytes of pseudo-random number, n bytes are copied from the pool to the output buffer; the remaining bytes are shifted so that they will be in the next output. The output blocks are hashed and inserted again into the pool to prevent exhaustion. Initial state of our F-PRNG is depicted in Figure 3.6 and generation procedure is in Figure 3.7. It should be noted that as the files on victim's computer gets more exclusive, i.e., different from other people's data, then the outputs of F-PRNG becomes harder to guess or reproduce after the attack as the plaintext versions of the files will be destroyed.

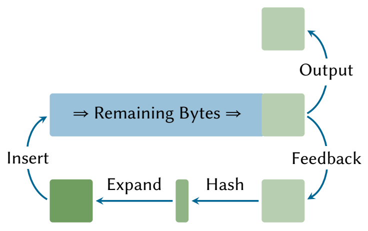


Figure 3.7: Output procedure of F-PRNG. As pseudo-random bytes are requested from the F-PRNG, the output buffer is filled with the requested amount. The remaining part of the pool is shifted accordingly. A copy of output bytes are hashed, expanded and inserted to the pool.

Expansion Process After providing the output bytes, that part is removed from the pool and the remaining bytes are shifted accordingly. This process shrinks the pool so that it exhausts in some finite time. To prevent this, we feed the pool with the pseudo-random numbers produced from the output that we call *expansion*. The method we use for expansion is similar to the approach used by Stark [64] and Eastlake [65], and described in Algorithm 1.

Asymmetric Key Pair Generation and Encryption

Ransomware needs to store the locally generated file encryption keys securely. Modern ransomware employs asymmetric algorithms for this task.

Algorithm 1 Expand a pseudo-random value to given length.

```
function EXPAND($input, n$)
    global $counter$                                   ▷ Pool keeps this counter.
    $\ell \leftarrow$ Length($input$)
    $max \leftarrow \lceil \frac{n}{\ell} \rceil$
    $i = 0$
    $output = [\ ]$
    for $i < max$ do
        $counter$ += 1
        $r$ = Hash($bytes \parallel counter$)   ▷ Generate pseudo-random chunk
        $output = output \parallel r$                ▷ and add to output.
    $output$ = Truncate($output, n$)   ▷ Output is truncated to n bytes.
    return $output$
```

Our imaginary ransomware also follows the same strategy. It employs the above F-PRNG to generate large primes to use in asymmetric algorithms[5], and to generate the padding values used for randomization of ciphertext.

5: Some real-world ransomware families, e.g., `CryptoLocker`, are designed to download asymmetric keys from C&C servers.

Evading Behavioral Analysis

Our second ransomware targets behavioral based defense systems that constantly monitor file system activity and look for anomalies. In particular, its objective is to encrypt files without triggering the indicators described in Section 3.1.

The presented variant, rather than using standard block ciphers, basically employs a Format Preserving Encryption (FPE) algorithm (see Figure 3.8). More specifically, the algorithm produces ciphertext which is a pure pseudo-random permutation of plaintext.

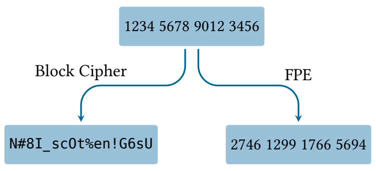


Figure 3.8: Contrary to traditional block ciphers, Format Preserving Encryption (FPE) creates a ciphertext that looks in the same format as the plaintext. For example, encryption of a 16-digit credit card number under FPE would be another 16-digit number.

Bypassing File-Type Checks

File-type probing is performed by inspecting the lead bytes of a file. Our ransomware therefore skips these bytes and starts encryption at a safe position. We determined this threshold empirically, testing over different file types including PDF, JPEG and DOCX. Our results shows that skipping the first 5120 bytes is sufficient for evading (**BA**-iii)[6].

6: Evading context-sensitive tests, which consider the entire file, would require a relatively complex algorithm to skip specific locations in a file.

Preventing Dissimilarity

Similarity of files is validated by comparing `sdhash` digests which produces a score between 0 and 100. According to the developers of `sdhash`, scores from 21 to 100 are considered as a strong indication of similarity [66]. In our experiments, comparing encrypted files with originals produces scores 0 or 1. However, we observed that partial encryption allows to obtain scores higher than 21, depending on the encryption ratio. (**BA**-ii) might set a lower threshold level, however, that would result in high false positive rates. Even in this case, tuning the encryption ratio would allow to keep this indicator silent. Figure 3.9 shows the partially encrypted files of different types and their corresponding similarity scores.

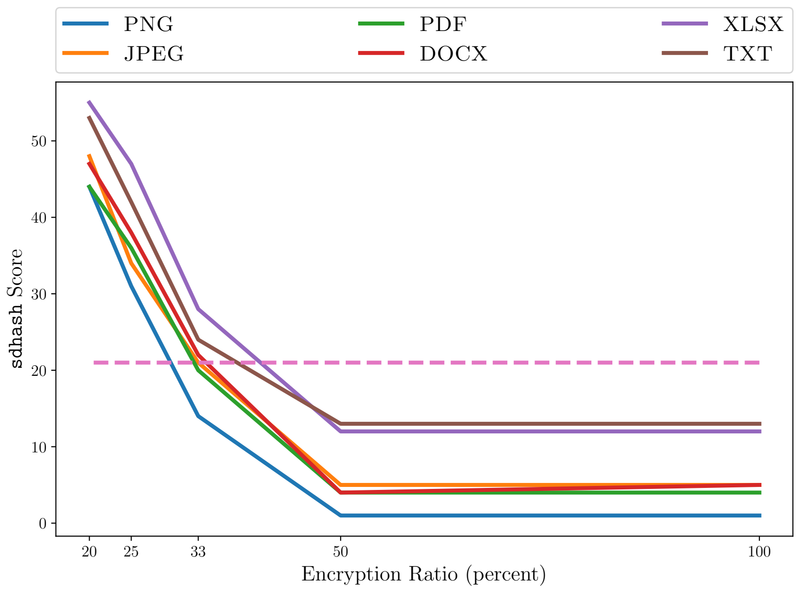


Figure 3.9: Average scores of `sdhash` comparison of partially encrypted file types. Scores above 21 (denoted by the dashed line) is considered as a strong indication similarity between compared file contents.

Evading Statistical Tests

(**BA**-i) measures the Shannon entropy of the files using Equation 3.1, before and after file-write operations, and monitors the increase. Standard encryption algorithms usually dramatically increase the file-entropy and so this is detectable. Instead, one might use a transposition style cipher to obfuscate files: the ransomware generates a pseudo-random permutation of the bytes of the plaintext blocks. If, as is commonly the case, the anti-ransomware tools use the measure Equation 3.1 then clearly permutation of the bytes leaves this invariant, and so this goes undetected.

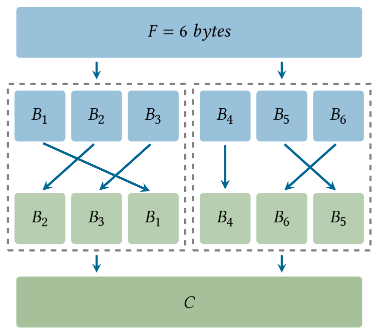


Figure 3.10: 3-byte permutation of a 6-byte array. Block size (which is set to 3 in this illustration) is adapted according to the buffer size of the target system. For example, we permute 64-byte of blocks in our implementation.

There are two obvious drawbacks with this approach: firstly such a transposition encryption is cryptographically rather weak, and secondly it only works for this particular measure of entropy of a string. A weak encryption may be good enough for the purposes of the ransomware, as long as the cost of cryptanalysis exceeds the ransom. Given that an easy counter is to use a different measure of entropy, or better still use more than one, this would not seem to be a long-term viable solution for the writers of ransomware.

Lastly, pure permutation technique also works against (**BA**-iv), the single indicator that DAD employs to detect encryption. DAD computes the sliding median of the χ^2 values of the last fifty write operations and compares this result to the threshold level $\alpha_{RW} = 0.05$. However, the χ^2 statistics (computed using Equation 3.2) remains constant under any permutation as the N_i values are not altered but rearranged. As a result, the permuted data does not fit the random distribution and (**BA**-iv) does not trigger the alarm.

3.4 Implementation

We have developed two prototypes in order to demonstrate the feasibility of the methods described in Section 3.3. Both programs are implemented in C# language targeting version 3.5 of .NET Framework. In addition, we ported the second prototype to Python (see Section 3.5).

The prototype which aims to bypass key-oriented defenses first enumerates the target files in the victim's computer. It uses an obfuscated SHA-256 function to compute hashes and the F-PRNG is initialized with 50 files' digests. This is the maximum capacity of the F-PRNG's pool which is implemented as a byte array. Our novel ransomware uses RSA algorithm for public key encryption. Once the F-PRNG is ready, two 1024 bit primes are generated, an RSA key pair is computed, and the private key is encrypted with the embedded master public key. Primality tests are performed using Miller-Rabin algorithm with the iteration count set to 3 as indicated in [67]. F-PRNG is also utilized to generate the padding values used for randomization of ciphertext.

The second prototype targets behavioral based approaches which monitors file system activities. It has two working modes: *partial* and *full* encryption. The former targets CryptoDrop and performs partial encryption and the latter fully obfuscates files. In our design, we set block size to $n = 64$, i.e., read 64 bytes, permute this block and overwrite the original data. Fisher-Yates algorithm [68] is utilized to permute the blocks (see Listing 3.1). We remark that, while executing Fisher-Yates algorithm, the required randomness is obtained from the CSPRNG API as behavioral analysis based systems do not control these.

Listing 3.1: Fisher-Yates algorithm implemented in Python.

```python
from random import randint

def fy_shuffle(a) -> None:
   n = len(a)
   for i in range(n-1, 0, -1):
      j = randint(0, i+1)
      a[i], a[j] = a[j], a[i]

a = [1, 2, 3, 4, 5, 6, 7, 8]
fy_shuffle(a)
print(a)

# [On Console]
# [4, 2, 5, 8, 3, 1, 6, 7]
```

Both of the prototypes contain only encryption routines, file I/O functions, and codes responsible for the key management tasks. As our main purpose is to show potential attacks and not to develop a fully functional ransomware, we deliberately omitted implementing all non-cryptographic functions, such as spreading over the network and deleting the Volume Shadow Copy Service (VSS) backups. Furthermore, our prototypes save a copy of encryption key in the same directory for each encrypted file to prevent accidental damages.

3.5 Experimental Results

In order to verify the feasibility of the methods described in Section 3.3, we tested our prototypes against ransomware defense systems in Table 3.1 that provides an implementation. In this regard, we conducted a series of experiments on PayBreak, DaD and CryptoDrop.

The test environment is prepared as follows. We created a Virtual Machine (VM) in VirtualBox[7] and performed a clean install of 32 bit version of Windows 7 OS. Next, we created 5 directories on user desktop and randomly placed decoy files therein. The decoy set contained 10 files with each of the extensions `.docx`, `.jpg`, `.pdf`, `.png`, `.txt` and `.xlsx`, making 60 in total. Before our experiments, we confirmed that the decoy files could be opened by the associated applications and were free of any corruption. Finally, we deactivated User Account Control (UAC) and

7: VirtualBox, https://www.virtualbox.org/.

Windows Defender to prevent interference, and took a snapshot of the test system.

We started experiments by testing the first prototype against PayBreak [8]. After running the executable of our first prototype, we observed that all decoy files were encrypted while the PayBreak was active. However, the log file of PayBreak did not contain any cryptographic material. As a result, we observed that our first prototype bypassed the software implementation of a key-escrow based defense system.

We continued our experiments with the behavioral analysis systems. We first tested the 32-bit version of DaD [9] against our second prototype. We activated DaD, executed the prototype and observed that all the decoy files were corrupted. Therefore, we conclude that our prototype could evade DaD.

Finally, we evaluated our prototype against CryptoDrop [10] as follows. Although we did not have an open source implementation of CryptoDrop, the mechanisms that [23] uses, i.e., `file` and `sdhash` tools are publicly available and installable on a Linux system. Moreover entropy changes can also be monitored using `ent`[11] tool. Therefore, we re-implemented our prototype in Python and run in partial encryption mode on a Linux system. We observed that `file` command reported that the original and encrypted files are of exactly same type. Moreover, all `sdhash` comparison scores were above 21 using %30 encryption. Finally, `ent` tool measured the partially-encrypted files have the same entropy with the original ones. Based on these results, we conclude that our prototype can bypass CryptoDrop.

We remark that partial encryption causes damage sufficient to make the files unusable (compare Figure 3.11 to Figure 3.12). In our experiments we observed that images could not be rendered and documents could not be read even with 20% encrypted files. Only exception is the TXT files that we could read the non-encrypted contents.

8: Compiled from source available at: https://www.github.com/BUseclab/paybreak.

9: Downloaded from http://people.rennes.inria.fr/Aurelien.Palisse/DaD.html.

10: This thesis analyzes the version of CryptoDrop described in [23].

11: ENT: A Pseudorandom Number Sequence Test Program, http://www.fourmilab.ch/random/.

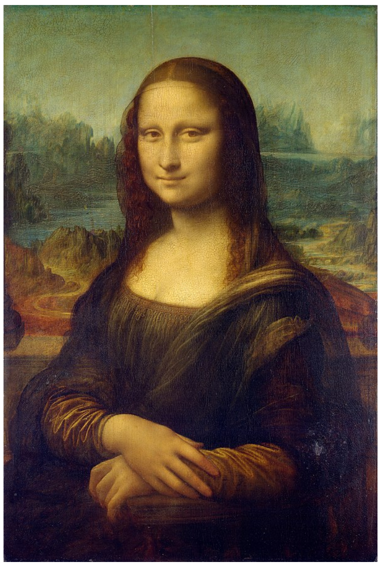


Figure 3.11: *Mona Lisa*, the masterpiece of Leonardo da Vinci.

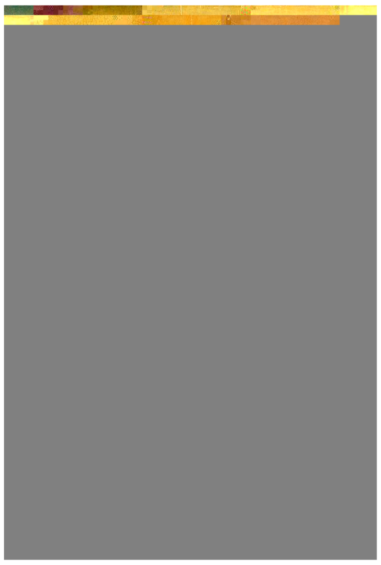


Figure 3.12: Partially encrypted Mona Lisa. Applied 20% permutation on the original image. The gray area is due to the inability of rendering the corrupted image.

3.6 Discussion

The purpose of this chapter is to warn the scientific community of forthcoming ransomware threats. By talking about how six cutting-edge anti-ransomware solutions —at the time of writing this thesis, implementing strategies of key escrow, placing backdoor in CSPRNG, and behavioral analysis are the most advanced strategies known against active ransomware samples— could be overthrown by smarter and more sophisticated malware, we hoped to have revealed what strategies those malware could trying to implement, so indicating where anti-ransomware engineers have to focus their efforts. Since it is believed that the ransomware threat will increase not in number of attacks but in sophistication, to keep anti-ransomware ideas ahead of time may be a game-changing factor.

That said, malware mitigation is an arms race and we expect new generations of ransomware coming soon with renovated energy and virulence, adapting their attack strategies to challenge current defenses. New variants of ransomware have been observed constantly during the last years. Those called *scareware* prefer to exploit people's psychology, threatening them into pay the ransom without, however, doing any serious encryption: despite deceitful they are technically benign applications. Others, however, will be variants of real cryptographic ransomware and able to overcome control and to encrypt a victim's files using strong encryption. A white paper by Symantec [3] reports that ransomware is becoming instrument for specialists and targeted attack groups, a weapon not only to extort money but to cover up other attacks and, when using strong encryption, used in fact as a disk wiper. It is to this second category that our research is dedicated. As security professionals we feel compelled to be prepared to face forthcoming threats thus to identify and anticipate potentially dangerous ransomware variants, and warn the scientific community about them.

On Deception-Based Ransomware Defense 4

In the context of ransomware, deception is primarily achieved by crafting artificial files, *decoys*, that the user is not supposed to see or access. In particular, by placing these files among real files of a user, a minefield-like area is established on the file-system. Whenever a process writes to a decoy file, it is immediately considered as a suspicious activity as any legitimate application would not access any of these files, and a predetermined response is taken.

One noteworthy aspect of deception-based ransomware detection is the lack of false positives (e.g., typically decoys are hidden from users to prevent users' mistakes). Another outstanding property of this approach is that it provides real-time detection with minimal overhead as no additional computation is involved, such as those performed by behavioral detection. However, the main issue of using decoys for ransomware detection is that if the strategy used to create them is weak, then ransomware can detect the presence of decoys and skip them while building the list of target files to encrypt. Therefore, to be effective, decoy files should mimic as closely as possible real-user files to deceive ransomware. The problem of building a robust decoy strategy shares similar properties with that of creating a realistic sandbox environment to perform dynamic analysis of malware [69]. In this scenario, the ransomware needs to be deceived that it is running on a real host while it is actually being run and monitored in an artificial environment prepared by the malware analyst. In fact, some ransomware tries to fingerprint the execution environment and look for traces of typical test systems, e.g., vendor ID of device drivers, and act as benign programs if they suspect of being monitored.

While decoy-aware ransomware is not an emerging threat yet, cybercriminals might turn their experience in fingerprinting sandbox environments for detecting decoy files. We envision this potential development and ask the following question to ourselves: *how secure are the current deception-based anti-ransomware systems?* (see Figure 4.1) It is crucial to find the answer of this question before such a development happens, as the damage of ransomware might be irreversible. Therefore, in this chapter, we take the task of analyzing the security of current decoy file strategies used to stop ransomware.

Figure 4.1: A decoy file should be indistinguishable from the genuine files of the user, like a land mine placed in the ground. Would an adversary step on the mine if it could be detected?

4.1 Decoy Files: The Theory

Decoy documents, sometimes called *honey files*, are fictitious files first introduced as a deception mechanism to detect unauthorized accesses to computer systems (see [70]). Their goal is twofold: (a) to attract the attention of intruders eager to steal information and lure them to access the files so revealing their presence *ex-ante*, and (b) to infiltrate bogus information that an insider entered in possession of the files may use eventually and somewhere. This signals an intruder's presence *ex-post.*

This second function, i.e., serving as a beacon of an intrusion activity after the intruder has left the system, seems less relevant to ransomware. Cryptographic ransomware operates without exfiltrating information and with the goal to block access to files in house. The use of decoy documents as alluring bait, instead, can be pivotal in revealing a ransomware's activity and in enabling strategies to minimize the damage.

Decoy files with this purpose have been proposed in academic literature (e.g., [33]) and also used in commercial products such as CryptoStopper [34], an anti-ransomware. In [32], the authors provide a solution that is pragmatically tailored to nullify the search and sorting strategies of eleven pieces of ransomware analyzed.

[33]: Mehnaz, Mudgerikar, and Bertino (2018), 'RWGuard: A Real-Time Detection System Against Cryptographic Ransomware'

[34]: WatchPoint Data (2018), *CryptoStopper*

[32]: Lee, Lee, and Hong (2017), 'How to Make Efficient Decoy Files for Ransomware Detection?'

Beyond the specific decoy strategies implemented by a few pieces of anti-ransomware, what are the qualities that a decoy file has to enjoy to be effective? Not being recognized as a decoy is surely one, but others may be relevant too. Other works have addressed the question in other domains, mainly in intrusion detection. In particular, in [71], Bowen *et al.* identify seven properties that a decoy file in its function of insider-trap has to enjoy, namely being: *believable*, that is recognized as if it were an authentic file[1]; *enticing*, that is alluring for the insider; *conspicuous*, that is, easily visible so to minimize the effort of the insider; *detectable* that is, serving well in detecting a malicious activity; *variability*, that is, not easily identifiable as bogus; *non-interference*, that is, not it making hard for honest users to recognize the non-decoy files; and *differentiable*, that is, easily discernible by honest users.

1: Juels and Rivest, who propose honeywords to detect a password leak, call it *flatness* [72].

Such properties, that in [71] come with probabilistic measures to quantify them, need of a little reinterpretation in the context of ransomware where the adversary is not a human masquerader aiming at finding and exfiltrating sensitive data. Properties such as "detectability", that in [71] is interpreted as hiding beacons (called decoy tokens) inside the files to trace them outside the system or injecting bogus information in a sort of counter-intelligence action, do not apply as is. Others, instead, do make perfect sense, like that of "being believable".

[71]: Bowen et al. (2009), 'Baiting inside attackers using decoy documents'

Quality Measures for a Decoy Strategy

Although understanding which properties a decoy file should have varies *per se*, in this chapter we prefer a pragmatic approach. We define two measures that are directly observable. The first is about the quality level of "deceptfullness" of a decoy strategy against a chosen ransomware. The second is a measure of its usability which directly links to the rate of false positives due to the activities of honest users.

We assume that the set of decoy files, D, is generated following a strategy $\mathtt{g}(\neg D, k)$ that can, but not necessarily, depend on some non-decoy files $\neg D$ (those which have to be protected) and some secret parameter k. We have no requirements on $\mathtt{g}$ but when talking of a "file" we mean content, name, and meta-data (e.g., be a hidden file, date of creation, last access, and so on), and also the directory structure that includes them. Function $\mathtt{g}$ can be static, that is defining D once and for all, or dynamic, that is changing D with time. It can be deterministic or randomized. We do not enter into the details of this procedure but a "good" $\mathtt{g}$ should make it hard for an adversary, A, to decide whether a given file f does belong to D.

Similarly, A's decision making can be based on a simple or complex selection strategy (i.e., it can be a deterministic search and sort, or can be a randomized search), however, what counts is that we can observe it. In other words, we can design experiments where A operates in an environment where it can access all files and where it is possible to find out what files A selects and encrypts and in which order. So, let $X^{\mathtt{g}}_{A}(f)$ be the random variable that returns the number of files that A encrypts before selecting and encrypting f, when A runs in a file system with files $F = \neg D \cup D$, and where D, the set of decoy files, is generated using $\mathtt{g}$. For $S \subseteq F$, we write $X^{\mathtt{g}}_{A}(S)$ to refer to the event $\min_{\forall f \in S}\{X^{\mathtt{g}}_{A}(f)\}$.

Definition 1 (Measure of Quality of a Decoy Strategy) *Let be A a ransomware, D the set of files generated by a decoy strategy $\mathtt{g}$, $F = D \cup \neg D$, $S \subseteq F$ a set of files, and n a natural number. The quality of a decoy strategy g is defined as follows:*

$$\Pr[\![X^{\mathtt{g}}_{A}(S) = n]\!] \tag{4.1}$$

It is the probability that A encrypts n other files before encrypting one in S.

When $S = D$, Equation 4.1 tells us the probability that A encrypts exactly n non-decoy files before encrypting a decoy file.

We can use Definition 1 in many ways. Intuitively, $\Pr[\![X^{\mathtt{g}}_{A}(D) = 0]\!]$ indicates the quality of a decoy generation strategy in fooling immediately A. A good decoy strategy should minimize $\Pr[\![X^{\mathtt{g}}_{A}(D) > 0]\!]$ that is the probability that a ransomware encrypts some good files before encrypting a decoy. It can be desirable to have a $\mathtt{g}$ that works more steadily against A: $\Pr[\![X^{\mathtt{g}}_{A}(\neg D) = n]\!]$ tells us the quality of a decoy strategy to keep ransomware busy for n decoy files before it eventually starts encrypting a good file. Intuitively, a good decoy strategy should maximize probability $\Pr[\![X^{\mathtt{g}}_{A}(\neg D) \geq n_0]\!]$, if n_0 is the minimal number of files that a certain anti-ransomware strategy needs before detecting that there is an illegitimate encryption in place. Often it can be sufficient to have $n_0 = 1$, but it may depend on the false positive rate of the anti-ransomware.

A strategy $\mathtt{g}$ that is also *usable* should non-interfere with the ability of a user U to recognize a non-decoy file. The experimental setting we propose to measure this quality assumes a random variable $Y^{\mathtt{g}}_{U}(S)$ that returns $\mathbf{1}$ when user U accesses to one of the files in a set S over a period of time in a working session (e.g., the time in-between two lock-screens); it returns $\mathbf{0}$ if U does not access to any file in S. We are interested in the following measure, that we call *confoundedness*, in which it indicates whether the user can get confused about his/her accessing non-decoy files.

Definition 2 (Measure of Confoundedness) *Let be U a user and D the set of decoy file generated according to a strategy $\mathtt{g}$, $F = D \cup \neg D$, and $S \subseteq F$ a set of files. Confoundedness is defined as follows:*

$$\Pr[\![Y^{\mathtt{g}}_{U}(S) = \mathbf{1}]\!] \tag{4.2}$$

It is the probability that U accesses a file in S within a working session.

Definition 2 is useful when we instantiate $S = D$. Intuitively, $\Pr[\![Y_U^{\mathtt{g}}(D) = \mathbf{1}]\!] = 0$ means that U never gets confused. Therefore, a usable decoy strategy should be able to keep $\Pr[\![Y_U^{\mathtt{g}}(D) = \mathbf{1}]\!]$ small, where small should be set according to empirical measure of the user experience, a value beyond which there is evidence that the user may switch off the decoy defence [73].

[73]: Balfanz et al. (2004), 'In search of usable security: five lessons from the field'

On the Theoretical Limits of Anti-Ransomware Decoy Strategies

The two measures we have defined in the previous section can be effectively computed once a decoy strategy has been defined and when either a threat model for A is set or when we have the possibility to observe A in execution.

We discuss here those measures in respect of a particular threat model for A. Bowen *et al.* [71] consider an adversary that is an insider, and define a "highly privileged" adversary as one having almost full control of the system, including knowing of the existence of a decoy strategy but without knowing it (i.e., A ignores $\mathtt{g}$). We assume at least the same "highly privileged" adversary. Under this condition, the adversary follows its own strategy to come out with a list of target files T from which to pick.

[71]: Bowen et al. (2009), 'Baiting inside attackers using decoy documents'

Even this "highly privileged" adversary may be however not as powerful as a ransomware can. In fact, due to its being able to run in a victim machine, ransomware may have a further weapon that instead is not available to a human insider: observing what files U accesses during a working session. Let us call this set of files $[F]_U$ and let us assume here that this is the set of files which U cares about: s/he would be willing to pay a ransom to have them back. Under this threat model we encounter a serious limitation of using decoy files as a general protection against ransomware. If $\mathtt{g}$ is perfectly usable, then its confoundedness is null, that is $\Pr[\![Y_U^{\mathtt{g}}(D) = \mathbf{1}]\!] = 0$. If A observed $[F]_U$, then A could simply choose among the files in $[F]_U$ to have a perfect strategy to avoid picking decoy files even without knowing how $\mathtt{g}$ works.

In general, however, $\Pr[\![Y_U^{\mathtt{g}}(D) = \mathbf{1}]\!] = p > 0$, which means that $[F]_U \cap D \neq \emptyset$. Assuming that A picks a target file in $[F]_U$ at random, it still has $\frac{|[F]_U \cap \neg D|}{|[F]_U|} \cdot p + (1 - p)$ chances to pick up a good file. Although a precise statistics can be computed only if all the parameters are set, if p is negligible it still gives a good chance of success to A. Instead if p is significant, that is, there is a high probability that U accesses a decoy file, it seems that it is better that U accesses as many decoy files as possible, which seems going against usability. Besides, we have to consider that A can also couple its random picking in $[F]_U$ with its own strategy to select files that are not decoy. This strategy is based on some intrinsic quality of the files, such as their names, extension, location, and this combination of strategies leads to an interesting theoretical question about what is the best game for A and for $\mathtt{g}$.

As far as we know such an A does not exists, and other strategies can be put in place to detect the presence of such an intrusive and curious ransomware, but our argument should be considered to raise awareness of what limits do exist when designing any `g`.

4.2 Anti-Ransomware Systems with Decoy Files

In this section, we give a brief description of some current anti-ransomware systems that uses deception-based strategy by implementing decoy files.

CryptoStopper

CryptoStopper is a commercial anti-ransomware solution developed by WatchPoint Data [34] and is advertised as "software to detect and stop ransomware". It places "randomly generated watcher files" in file system to detect ransomware. According to WatchPoint Data, the average time for CryptoStopper to detect a ransomware is 9 seconds.

[34]: WatchPoint Data (2018), *CryptoStopper*

In the case that malicious activity is detected, i.e., a process tries to write to a decoy file, CryptoStopper alerts the system administrator and the infected host is shut down. Furthermore, other computers at the network are notified so that (if they are running CryptoStopper) they drop packages coming from the infected host, isolating that machine from the network. In this regard, CryptoStopper can also be viewed as a local threat intelligence system that can protect the network from a zero-day ransomware with minimal loss.

RWGuard

RWGuard [33] unifies techniques from previous proposals in a single tool to mitigate cryptographic ransomware. To detect ransomware, RWGuard comprises dedicated modules to (i) check if a decoy file is modified; (ii) monitor process behaviours; (iii) identify abnormal file changes; (iv) classify user's cryptographic activity; and (v) control built-in cryptographic API.

[33]: Mehnaz, Mudgerikar, and Bertino (2018), 'RWGuard: A Real-Time Detection System Against Cryptographic Ransomware'

Decoy generator tool in RWGuard uses the original files of the user. The authors state that the names of decoy files are generated similar to the genuine user files and in a way that decoys can be clearly identified, though, the exact procedure is not described. The number of decoy files is determined by the user for each directory. The extension list of decoy files are static (`.txt`, `.doc`, `.pdf`, `.ppt`, and `.xls`) and their contents are created by copying from user's genuine files. The sizes of decoy files are randomly chosen from a range based on the sizes of user's genuine files while the total size of decoy files is limited to 5% of user's genuine files.

R-Locker

[74]: Gómez-Hernández, Álvarez-González, and García-Teodoro (2018), 'R-Locker: Thwarting ransomware action through a honeyfile-based approach'

To detect ransomware, R-Locker [74] employs decoy files but in a slightly different manner. The proposed approach is to create a central decoy file in user's home directory, which is actually a First-In First-Out (FIFO) special file, i.e., a named pipe. Next, R-Locker writes a few bytes to this FIFO file, which will not be read until a process accesses to the pipe. Consequently, any process trying to read this pipe will trigger the protection module and will be detected. Authors suggest to place multiple symbolic links pointing to the central decoy file in various locations on the file system to decrease the time to detect ransomware.

In contrast to the other anti-ransomware systems, R-Locker interprets any read access to decoy files as ransomware activity. The false-positive rate of this approach, i.e., the frequency of occurrences of read operations initiated by a legitimate process like background service or a system daemon, is not evaluated, though.

Decoy Generation Strategy of Lee et al.

[32]: Lee, Lee, and Hong (2017), 'How to Make Efficient Decoy Files for Ransomware Detection?'

In [32], Lee *et al.* reverse engineered 11 cryptographic ransomware samples from different families and analyzed their file system traverse patterns. Based on this analysis, the authors developed a method which generates decoy files in directories that the samples they analyze starts to traverse from. The authors also found that these samples sort files alphabetically. Based on this observation, the proposed method creates two decoys: one with a file name which comes first in normal ordering, and another decoy that comes first in the reverse order, to nullify both ordering strategies.

In addition to the current ransomware, Lee *et al.* also attempt to anticipate the possible evasion methods that may come up in the future. In this regard, the authors extend their algorithm by taking into account the alternative orderings based on file size and access time. Furthermore, the authors mention the case that ransomware orders files randomly and propose "monitoring random function calls to detect the ransomware which traverse in random order". However, we were not able to understand how exactly the proposed algorithm works.

Other Closely Related Works

In the previous subsections, we have investigated the related work involved with the findings of this research. In this subsection, we summarize other works related to the use of decoys in ransomware mitigation.

[75]: Moore (2016), 'Detecting Ransomware with Honeypot Techniques'

One of the first honeypot systems against ransomware is proposed by Moore in [75], which tracks the number of files accessed on specified directories. The system implements a hierarchical multi-tier response model. Depending on the number of files accessed, the level of severity is determined and the corresponding countermeasure is applied.

[76]: Moussaileb et al. (2018), 'Ransomware's Early Mitigation Mechanisms'

Moussaileb *et al.* in [76] developed a post-mortem analysis system that detects ransomware activity using machine learning techniques. In their analysis, authors investigated the directory traverse patterns of processes and classified ransomware based on traversal traces in decoy directories.

Feng *et al.* in [77] intercepts `FindFirstFile` and `FindNextFile` API to manipulate file system traverse of processes so that whenever a process looks for a file, it is first served with a decoy file. Once the process finishes its task on the decoy file, the monitoring module of the system perform checks employed in behavioral analysis systems. After the checks, a process that shows malicious traits is terminated.

[77]: Feng, Liu, and Liu (2017), 'Poster: A New Approach to Detecting Ransomware with Deception'

4.3 Decoy-aware Ransomware

In the previous section we discussed some key elements of the strategies of a few anti-ransomware systems. They can be considered instances of what we called `g`. We now imagine a few anti-decoy strategies for a ransomware, which then becomes a *decoy-aware ransomware*. Such strategies can be either to black-list files that the ransomware considers decoy, thus not to be encrypted, or to list files that it labels as user files, thus to be encrypted. In particular, we describe how ransomware can detect decoys by relying on heuristics, for instance the presence of zero-filled files (Section 4.3) and on statistical methods (Section 2). Then we describe how ransomware can find out the files accessed by users which are quite likely non-decoy files (Section 2).

Detecting Static Decoys through Heuristics

A ransomware can look for patterns that are indicative of decoy files generated by some (weak) strategy, such as files that are hidden or filled with empty values, or where the creation date and content include a static pattern which can be discovered by the attackers.

Similarly to the case of *anti-analysis* techniques (e.g., anti-sandbox/anti-debugging), ransomware authors can first create a database of fingerprint-based decoy checks to be included in future versions of ransomware. This set of fingerprinting checks can be then performed at run-time when scanning the victim computer to create the list of target files to encrypt. Note that, differently from the case of anti-analysis, in which a malware typically stops performing any malicious actions if it detects signs of an analysis environment, ransomware does not stop performing its malicious operations when decoys files are detected but simply excludes them from the target files.

One fact must be observed. A false positive is less troublesome in the case of a ransomware's anti-decoy detection with respect to anti-analysis. If the ransomware skips a user file mistakenly believed to be a decoy, this has a little impact on the overall strategy of the ransomware. The ransomware still encrypts several other files, and the request for ransom still holds.

In the following, we describe two heuristics for decoys files. For the first (see Algorithm 2) decoys are files that are hidden and empty. For the second (see Algorithm 3) decoys are non-regular files, such as symbolic links or named-pipes.

Algorithm 2 Collect files that are not hidden or filled with zero value.

```
function Collect($path$)                          ▷ Directory of files to scan.
    $FileList \leftarrow$ EnumerateFiles($path$)
    $GenuineList \leftarrow \emptyset$
    for all $f \in FileList$ do
        if IsHidden($f$) then
            $allNull \leftarrow$ True
            while not EOF do
                $b \leftarrow$ f.ReadByte()
                if $b \neq 0$ then
                    $allNull \leftarrow$ False
                    break                 ▷ $f$ might not be decoy, try next file.
            if $allNull$ = True then
                $GenuineList \leftarrow GenuineList \cup \{f\}$
        else
            $GenuineList \leftarrow GenuineList \cup \{f\}$
    return $GenuineList$
```

Algorithm 3 Collect files on Ext4 FS that are not a FIFO or a symbolic link.

```
function Collect($path$)                          ▷ Directory of files to scan.
    $FileList \leftarrow$ EnumerateFiles($path$)
    $GenuineList \leftarrow \emptyset$
    for all $f \in FileList$ do
        if IsPipe($f$) then
            continue                                          ▷ Skip pipes,
        else if IsSymbolicLink($f$) then
            continue                                ▷ and symbolic links.
        else
            $GenuineList \leftarrow GenuineList \cup \{f\}$
    return $GenuineList$
```

[74]: Gómez-Hernández, Álvarez-González, and García-Teodoro (2018), 'R-Locker: Thwarting ransomware action through a honeyfile-based approach'

With the exception of [74], all deception-based anti-ransomware systems that we know trigger protection when a decoy file is modified or deleted but not when the file is only read. Thus Algorithm 2 and Algorithm 3 can work mostly undetected. And, since modern file systems store a file's contents and metadata separately (see Section 2), Algorithm 3 might even be able to work without reading the actual files but only their metadata, reaching full stealthiness. For example, on Linux OS, Algorithm 3 can obtain all the required information by calling `readdir()`[2] function.

2: For more details, see the documentation at http://man7.org/linux/man-pages/man3/readdir.3.html.

Distinguishing Decoys Using Statistical Methods

Let us first briefly recall some technical details of the file storage on Windows OS to understand file attributes and metadata.

On modern versions of Windows platform, New Technology File System (NTFS) is the default file system for controlling the storage and retrieve of data on the disk. In NTFS, all data are stored in files. In addition to the data stored as regular files, the internal data for structuring the file system are also stored as files. These auxiliary files are called *metadata* files. Among the metadata files in NTFS, the most important is the MFT, which contains the description of all files on a disk volume. This information includes the file name, time stamps, e.g., date created and date last accessed, security identifiers and file attributes, e.g., hidden and read-only. Table 4.1 shows a list of selected attributes for files on NTFS volumes.

Table 4.1: Selected attributes for NTFS files. The complete list is available at [78].

Attribute Name	Description
`$STANDARD_INFORMATION`	File attributes such as read-only, archive, and so on; time stamps, including when the file was created or last modified.
`$FILE_NAME`	File name in Unicode 1.0 characters.
`$DATA`	The contents of the file.
`$ATTRIBUTE_LIST`	A list of the attributes that make up the file and the file record number of the MFT entry where each attribute is located.

Therefore, since the decoy files are not supposed to be accessed by users, its reasonable to expect that their access pattern shows difference from genuine files. Under NTFS, it is possible to obtain certain parts of information about access patterns, for example, `$STANDARD_INFORMATION` attribute stores last modification time. Furthermore, one can access file creation time along with other file attributes. This basic information, statistically analyzed, can be used to distinguish decoy and genuine files.

In fact, this observation was made by Rowe [79] in 2006. Rowe developed a tool which computes statistics on a file system and compares this data to that of a typical real system. If there exist a significant discrepancy in the two statistics, this suggests the existence of a deception-based defense. The proposed tool computed 36 metrics for each directory using features based on file attributes, including (i) file name; (ii) file size; (iii) file type; and (iv) last modification time. Furthermore, these 36 metrics are augmented with their standard deviations, resulting in a feature vector of 72 metrics. For a file system i, let μ_{ik} and σ_{ik} denote the mean and standard error of metric k. Rowe assumed that all metrics have equal weights and used the following formula to compute the difference between systems i and j:

[79]: Rowe (2006), 'Measuring the Effectiveness of Honeypot Counter-Counterdeception'

$$s_{ij} = \left(\frac{1}{72}\right) \sum_{k=0}^{35} \left[\frac{|\mu_{ik} - \mu_{jk}|}{\sqrt{\sigma_{ik}^2 + \sigma_{jk}^2}} + \frac{|\sigma_{ik} - \sigma_{jk}|}{2\sigma_k} \right] \tag{4.3}$$

According to results of Rowe's experiments, Equation 4.3 excels in finding discrepancies based on time & date information.

The feasibility of this technique, among with its efficiency makes it a valuable tool for a decoy-aware ransomware. Unless the anti-ransomware system updates the decoy files in such a way to mimic the user behaviour, Rowe's results shows that statistical techniques increase the chances of attackers against decoy-based defenses. We elaborate on this issue more in Section 4.5.

Monitoring User to Reveal Non-decoy Files

An anti-ransomware system that uses decoy files is supposed to be designed in such a way to let legitimate users either be able to differentiate between genuine and decoy files, or not to be able access decoy files for instance by hiding them. Either way, the goal is to prevent the user from accessing a decoy file.

Relying on this consideration, a decoy-aware ransomware can obtain a list of genuine files by monitoring the user activity. In a metaphor, by following the user's steps, the ransomware can pass unharmed through the minefield of decoys. We imagine two of such decoy-aware ransomware strategies:

Algorithm 4 Monitor User.

```
1: function MONITOR
2:     $Exp \leftarrow$ FindProc($Explorer$)
3:     InjectProc($Exp, SpyModule$)
4:     $GenList \leftarrow \emptyset$
5:     while true do
6:         $f \leftarrow$ Listen($SpyModule$)
7:         $GenList \leftarrow GenList \cup \{f\}$
8:     return $GenList$
```

Algorithm 5 Replace `WriteFile`.

```
1: function REPLACE
2:     $PList \leftarrow$ EnumAllProc()
3:     for all $p \in pList$ do
4:         InjectProc($p, InterceptMod$)
5:         $wf \leftarrow$GetFnAddr(WriteFile)
6:         if $wf \neq$ NULL then
7:             Replace($wf, encFile$)
8:     return Success
```

- (see Algorithm 4): inject a spy module into `Explorer.exe` to monitor which files are accessed by user applications Ransomware can further compute the hash of the file at first access time and check it later for changes to detect modifications – this might be a sign of a "valuable" file (though, not always this property holds: e.g. pictures are rarely changed, and they are very valuable for ransomware).
- (see Algorithm 5): Enumerate all processes and inject an interceptor module which hooks `WriteFile` API Replace the `WriteFile` API with the encryption routines (however, this strategy also requires the ransomware to keep information about which parts of the files have been overwritten to be able to properly decrypt it later).

Decoy-aware ransomware implementing Algorithm 4 and Algorithm 5 can run in user-space, i.e., no kernel-mode component is required; so, the ransomware would typically have the sufficient privileges to run.

4.4 Experiments and Quality Measures

We demonstrate the feasibility of our speculated decoy-aware ransomware, and we measure against it, the quality of decoy of CryptoStopper, the only anti-ransomware we could have access among the ones we described in Section 4.2, and of DecoyUpdater, a proof-of-concept of an anti-ransomware that implements the mitigation strategy that we described in Section 4.5.

Revealing Static Decoys

To demonstrate the feasibility of the avoiding static decoys, we have developed a prototype implementing Algorithm 2. We conducted the experiments on a clean install of Windows 10 (version 1809) virtual machine (VM) running atop VMware Fusion. In the experiments, we populate the VM with 30 files, namely 5 from each of the following file types that are typically selected by ransomware: `.txt`, `.jpg`, `.png`, `.docx`, `.xlsx` and `.pdf`. These 30 files are placed in user's directories targeted by ransomware, including Desktop and Documents. Once the artificial environment is ready, we have tested it on the latest version CryptoStopper at the time of writing.

We implemented Algorithm 2 in C# language and run it on the test system. As shown in Figure 4.2, our prototype successfully identified all the 25 decoy files generated by CryptoStopper and skipped them while building the list of targeted files to encrypt.

AntiStaticDecoy Console

```
AntiStaticDecoy Prototype
Target directory: C:\Users\RWTest\Desktop

[GENUINE] C:\Users\RWTest\Desktop\accesscontrol.jpg
[GENUINE] C:\Users\RWTest\Desktop\AccountSummary.xlsx
...
[GENUINE] C:\Users\RWTest\Desktop\weakprng.png
[GENUINE] C:\Users\RWTest\Desktop\WeeklySchedule.txt

Total number of genuine files: 30

[DECOY] C:\Users\RWTest\Desktop\Parlay Permit\Add Resolve.docx
[DECOY] C:\Users\RWTest\Desktop\Parlay Permit\Backup Convert.docx
...
[DECOY] C:\Users\RWTest\Desktop\Parlay Permit\Unregister Test.avi
[DECOY] C:\Users\RWTest\Desktop\Parlay Permit\Use Initialize.pptx

Total number of decoy files: 25

Proceed with encryption? y/n
```

Figure 4.2: Console output of the prototype of Algorithm 2.

The fact that CryptoStopper (*CS*) relies on a static strategy means that the set of decoy files has features that do not change in time. Thus, we can safely infer from our experimental result that an estimation for our measure of quality of decoy strategy is $\Pr\llbracket X^{CS}_{Alg_1}(\neg D) > 0\rrbracket = 0$ and $\Pr\llbracket X^{CS}_{Alg_1}(D) > 0\rrbracket = 1$. Actually, we estimate, $\Pr\llbracket X^{CS}_{Alg_1}(D) > n\rrbracket = 1$ for all $0 \leq n \leq |F|$. Thus our version of a hypothetical ransomware outsmarts CryptoStopper. In defence of CryptoStopper we have to say that we tested it against a potential but a non-existing ransomware variant. So the lesson we can learn from this experiment is not precisely on CryptoStopper, but rather on a strategy that generates decoy files with fixed and communal properties. Knowing it, a ransomware designer can easily implement a counter-measure that sieves those files from the rest.

Decoy-aware ransomware implementing Algorithm 3 were supposed to be tested against the other anti-ransomware systems in Section 4.2; we requested the prototypes of [33] and of [74] to conduct experiments but not received any response from the authors yet. The method proposed in [32] is not published, so it is unavailable.

[33]: Mehnaz, Mudgerikar, and Bertino (2018), 'RWGuard: A Real-Time Detection System Against Cryptographic Ransomware'

[74]: Gómez-Hernández, Álvarez-González, and García-Teodoro (2018), 'R-Locker: Thwarting ransomware action through a honeyfile-based approach'

[32]: Lee, Lee, and Hong (2017), 'How to Make Efficient Decoy Files for Ransomware Detection?'

Revealing non-Decoy Files by Monitoring Users

To demonstrate the feasibility of our hypothetical ransomware monitors users, we implemented Spy and Replace. They realize Algorithm 4 and Algorithm 5, respectively.

Spy is written in C# and uses `FileSystemWatcher`. The target directory to watch for events is set to `%USERPROFILE%\Desktop`. Spy implements `OnChanged` and `OnRenamed` event handlers to receive file change notifications, and `OnCreate` for watching new files.

Replace is implemented as a Dynamic Link Library (DLL) module using C++ language, and injected into the target process by calling `CreateRemoteThread`. Once the DLL is loaded into the target application's memory area, all hooking operations are performed using Detours library [80] from Microsoft Research. After loading the malicious DLL, Replace hooks `WriteFile` API and whenever `WriteFile` is called, it invokes `Fake_WriteFile` that encrypts the whole content of the file using `CryptEncrypt` with a hardcoded key[3].

[80]: Hunt and Brubacher (1999), 'Detours: Binary Interception of Win32 Functions'

3: For the sake of proof-of-concept: a real ransomware would use a strong key-management strategy.

The experiments were conducted on a similar setup environment as the previous one, namely a Windows 10 VM running atop VMware Fusion with 30 user files created and placed similarly as before. In addition to these user files, we have also added two decoy files from each file type.

```
Spy Console

Target Directory: C:\Users\RWTest\Desktop

20:58:07.814 CHANGED C:\Users\RWTest\Desktop\MyPasswords.txt
20:58:13.814 CHANGED C:\Users\RWTest\Desktop\TermProject.doc
20:58:21.002 CHANGED C:\Users\RWTest\Desktop\Essay.doc
20:58:30.439 CHANGED C:\Users\RWTest\Desktop\MyNotes.txt
20:58:42.626 CHANGED C:\Users\RWTest\Desktop\AccountSummary.xls
20:58:46.955 CHANGED C:\Users\RWTest\Desktop\Costs.xls
```

Figure 4.3: Console output of Spy on unprotected system.

During the experiments, we first run Spy, and then use various applications to open and read the content of all of the 30 user files, by writing at various time intervals to the `.txt`, `.doc` and `.xls` files only. As shown in Figure 4.3, Spy is able to successfully observe all the user files that have been modified.

Thus, speculatively Spy (as well as Replace) is able to nullify existing decoy methods, be this one CryptoStopper or one of the solutions we described in Section 4.2.

The only significant comparison is against the anti-ransomware that employs (**F**1) and (**F**2) (see Section 4.5) in a decoy-file based defense system. This is the prototype we called DecoyUpdater[4] (DU). It should be noted that our aim is not to provide a full-fledged deception system. Rather, we attempt to evaluate our technique and prove the validity and efficiency of the underlying idea (see Chapter 2).We have developed DecoyUpdater in C# language. For ease of implementation, we have used the `System.IO.FileSystemWatcher` class, as it is very useful for monitoring file system events, such as for opening/deleting/renaming files and directories or detecting changes in file contents.

4: Available under GPLv3 at `https://github.com/ziyagenc/decoy-updater`.

We have started DecoyUpdater and have repeated the same previous actions on files (namely, reading and writing), seen in Figure 4.4. This time, however, event logs in the Spy also show the file activities performed on the decoys, as in Figure 4.5. Since the logic behind Spy is bound to the OS and does not depend on other factors, from the only experiment we have run, we obtain $\Pr[\![X^{DU}_{Spy}(D) > 0]\!] < 1$ and $\Pr[\![X^{DU}_{Spy}(\neg D) > 0]\!] > 0$. A more precise estimate requires to equip *Spy* with a decisional strategy over the collected files, and we leave this a future work.

```
DecoyUpdater Console

Target Directory: C:\RWTest\RWTest\Desktop

07:15:31.608 CHANGED C:\RWTest\RWTest\Desktop\ToDoList.txt
07:15:33.616 UPDATED Decoy file: Addresses.txt
07:15:50.790 CHANGED C:\RWTest\RWTest\Desktop\MyPasswords.txt
07:15:51.792 UPDATED Decoy file: PhoneNumbers.txt
07:15:58.641 CHANGED C:\RWTest\RWTest\Desktop\MyNotes.txt
07:15:58.643 UPDATED Decoy file: Addresses.txt
```

Figure 4.4: Console output of DecoyUpdater while Spy is active. The decoy files `Addresses.txt` and `PhoneNumbers.txt` are randomly picked and updated after a random delay of 5 seconds maximum.

```
Spy Console

Target Directory: C:\Users\RWTest\Desktop

07:15:31.608 CHANGED C:\Users\RWTest\Desktop\ToDoList.txt
07:15:33.616 CHANGED C:\Users\RWTest\Desktop\Addresses.txt
07:15:50.790 CHANGED C:\Users\RWTest\Desktop\MyPasswords.txt
07:15:51.792 CHANGED C:\Users\RWTest\Desktop\PhoneNumbers.txt
07:15:58.641 CHANGED C:\Users\RWTest\Desktop\MyNotes.txt
07:15:58.643 CHANGED C:\Users\RWTest\Desktop\Addresses.txt
```

Figure 4.5: Console output of Spy while DecoyUpdater is active. Note that the list contains the decoy files `Addresses.txt` and `PhoneNumbers.txt`.

As the final set of experiments, first we have executed a helper program to inject the Replace module into a target application, namely `Notepad.exe`. Using this target application, we have opened all the `.txt` files and added some random text into all of them, and saved them. After this operation, the `.txt` files were encrypted by Replace crypto-module. Second, we have activated DecoyUpdater and repeated the same steps in the previous experiment. In this scenario, at each try, DecoyUpdater's operations were intercepted by the Replace module. However, Replace's activities has been successfully reported[5] by DecoyUpdater in the logs, which is shown in Figure 4.6. Again, since the strategy's logic depends only on the OS, we can, from only one experiment, estimate that $\Pr[\![X^{DU}_{Replace}(D) > 0]\!] < 1$ and $\Pr[\![X^{DU}_{Replace}(\neg D) > 0]\!] > 0$. DecoyUpdater strategy has the potential to become robust deception-based anti-ransomware system, but demonstrating this claim is left for the future.

5: Due to the limited capability of `System.IO.FileSystemWatcher` class, we could observe the malicious activity, yet we were not able to identify the process ID of Replace and terminate it. That would be possible with developing a file system mini-filter, which is an implementation effort.

```
DecoyUpdater Console

Decoy Updater v1.0

Status: ACTIVE
Target Directory: C:\Users\RWTest\Desktop

22:00:27.553 CHANGED C:\Users\RWTest\Desktop\MyPasswords.txt
22:00:30.585 UPDATED Decoy file: PhoneNumbers.txt
22:00:30.585 WARNING Unexpected Write: PhoneNumbers.txt
```

Figure 4.6: Console output of DecoyUpdater while Replace is injected into Notepad.exe and active. The logs shows that malicious activity on the decoy file `PhoneNumbers.txt` is detected.

4.5 Discussion: The Endless Battle

History suggests that the malware mitigation is a multifaceted combat where the cyber-criminals constantly searches for a hole in the battlefronts. It is not a secret that, to achieve their nefarious aims, ransomware authors also acquire new techniques to exploit the limitations of defense systems. A good deception-based anti-ransomware strategy, say $\mathtt{g}$, we argued in Section 4.1, should be, at least, such that to maximize the probability for a ransomware A to encrypt first any decoy files (i.e., $\Pr[\![X_A^{\mathtt{g}}(\neg D) > 0]\!]$); similarly, it should also minimize the probability it starts encrypting some genuine files first (i.e., $\Pr[\![X_A^{\mathtt{g}}(D) > 0]\!]$). Such probabilities can be enriched to consider for how long (i.e., for how many encrypted files) g is capable of keeping the ransomware in check.

In Section 4.4 we give an experimental estimation of those measures of quality for some of the deception-based anti-ransomware strategies—those we could get access to plus two we designed ourselves and that we describe in this section—against the ransomware strategies that we have imagined to exist and that we implemented and run. Here, we discuss how they can minimize the damage of a ransomware attack.

To begin with, as we argued in Section 4.3, the static decoy files can be practically discovered by a decoy-aware ransomware and therefore should be avoided. In order to prevent fingerprinting of the decoys employed, the defense system should include randomness in the decoy generation procedure. Note that this property should not be understood as filling the decoy file with CSPRNG outputs as the ransomware can also detect the unusually high entropy in the file content. Though, we cannot reach an ultimate decision in such case; ransomware may interpret the file as a trap and skip or a valuable data, e.g., encrypted key vault, and attack.

As we argued in Section 2, ransomware can obtain crucial information from the metadata in a file system and use statistical techniques which might enable to unveil decoy files. RWGuard updates the decoy files *periodically* to mitigate this potential attack, however, we believe any update pattern would result in a discrepancy. Reasonably, the best protection level looks like reflecting user behaviour on decoys. We left such a decoy system to be realized in a future work. Randomness is also vital when updating the decoy files (see later). That said, an under-studied aspect of decoy files is the header-extension relation. An inconsistency in header bytes and file extension might make a decoy-aware ransomware suspicious, therefore, these two should be coherent. Moreover, the decoy updater process should be careful if a new content is added to a file randomly, and target the body of the decoy file. This can be usually achieved by skipping the first few bytes of the decoy file.

Decoy-aware ransomware that observes user-behaviour, whose existence we speculate in Section 4.3, could be quite hard to beat. As a mitigation strategy, an anti-ransomware could add the following functionalities to current decoy systems:

(F1) **Add noise to user activity** by emulating user's opening a decoy file so that the spy module adds a decoy file to *GenList*. If the file accessed by the user is modified, update a decoy file to mimic the user.

(**F**2) **Verify the data written to decoy file** to check if the decoy updater is compromised.

To bypass these strategies, a ransomware may ignore a series of user activities happening in a short time-frame. Therefore, to obfuscate the functionality of item (**F**1), a decoy updater may choose to delay the update process for a random time period or – ideally – according to user's access pattern. Predetermined update patterns may also be identified by attackers. Therefore, item (**F**2) must use randomized data while updating decoy files. On the other hand, item (**F**2) should also avoid writing to the file headers so that file's magic value does not conflict with its extension. Although it might be unfeasible to locate the process that initiated the attack, a protection system might suspend all file system activity when an inconsistency is reported by item (**F**2). In Section 4.4 we build such a anti-ransomware based on these ideas, called *DecoyUpdater*, and estimate its quality.

Another under-studied aspect of anti-ransomware solutions employing decoy files is the *usability*. This topic deserves an independent research, but we would like point out two issues. The placement of decoys are studied fairly well in the past; however, the effects to user's daily workflow needs more research. For example, if the decoy files are generated with the hidden attribute, it would be safe for ransomware to attack only visible files. This may suggest to generate the decoys as visible files, and therefore at least an estimation of our measure of confoundedness, $\Pr[\![Y_U^{\mathsf{g}}(D) = \mathbf{1}]\!]$ is required.

The number of decoys has also another significance: to evade ordering strategies described in Section 4.2, decoy-aware ransomware might utilize a random ordering. In this case, obviously, the more decoy files, the faster detection speed. It should be noted that, the number of decoys may not be useful against selective attacks, though.

Lastly, deception-based defense systems are highly linked to the security of the host OS. If for example, ransomware can write to MBR and reboots the target machine, it might be able to load a malicious kernel and encrypt the files, as `NotPetya` does. However, this is rather a generic issue about runtime protection systems and applies to the most of the other anti-ransomware solutions.

Vulnerability Analysis of Real-World Systems 5

To protect IT assets, distinct classes of basic security practices are often provided to the end users depending on their usage scenario. For instance, home users are instructed to always update their OS and applications; corporate administrators are required to employ some form of user training to teach users, e.g., how not to click on e-mails that look suspicious; organizations are recommended to use firewalls to protect their networks from remote attackers. However, it is often the case that the first security recommendation given to all classes of users is to install an antivirus (AV) on their devices. In fact, AVs are believed to be one of the best protection solutions, specifically against malware. They are installed in most user computers and companies, and are implicitly trusted by most users, and are part of the trusted computing base[1].

1: Most AVs require kernel-level privileges to perform some of their operations.

It goes without saying that, while AVs do offer protection, they cannot catch all malware. Not only there might be missing signatures in their database [81], but over the years malware authors have spent great effort in trying to evade AVs detection, e.g., through obfuscation and polymorphism [82] or evasion [83], or by disabling or crashing the AV [84, 85]. AVs and malware are engaged in a *cat-and-mouse* game: attacks based on polymorphism are typically mitigated by some form of anomaly or behavioral detection [86, 87] while evasion attacks are mitigated by making the AV more difficult to exploit, such as through OS protection and standard binary integrity protection techniques [88]. The battle continues on, as now malware can try and bypass AV behavioral detection using, for instance, adversarial inputs [89], and AVs will incorporate robust mechanisms to mitigate the effects of these inputs [90, 91].

In this *cat-and-mouse* game, one party tries to anticipate the move of the other. We believe AVs should be the party more involved in this thinking-ahead, for instance by questioning whether certain principles, or certain best practices on which their defences rely upon, are valid, and under which assumptions and limitations they are so. Taking this stand, let us consider the best practice of using a white-list instead of a black-list. The advantage of white-lists has been largely discussed elsewhere, but what we question here is whether AVs, and OS, make their security dependent too much on them by assuming that white-listed built-in applications of OS, like `Notepad` and `Paint`, can never do any harm. To test this hypothesis, we create a malware that can control those white-listed applications like puppets by instructing them to perform malicious operations. Then we verify whether, dressed this way, our malware can bypass all the defences that AVs and OS put in place to protect files, e.g., have them in the so called *Protected Folders*. It turns out that we can, and we named this attack *Cut-and-Mouse*.

In our second hypothesis, we question whether AVs go one step further, and assume that users have the choice to decide whether to set the offered protections off and on, without considering whether it is really the user doing so, or someone (or something) just simulating the user's behaviour.

Thus we tested whether off-the-shelf AVs can be disabled (i.e., turned off or freeze) by a malware that mimics user inputs (through synthesized keyboard and mouse events). We expected that AVs would have enforced some forms of integrity and authentication checks on inter-process communications, and user access control to verify the legitimacy of the received inputs. It turns out that a great deal of them do not. Our attack, named *Ghost Control*, managed to disable several AVs by spoofing requests to their main graphical interface.

Problem Statement: We claim that the following problems exist in current malware mitigation:

(**P**-i) Several AV programs contain a critical flaw that allows unauthorized agents to turn off their protection features. In detail, the real-time scanning service of some AVs can be disabled by malware. This will make victims exposed to several kinds of cyber-threats, especially those originated from malware.

(**P**-ii) *Protected Folders* solution provided by AV vendors suffers from design weaknesses. In fact, a small set of whitelisted applications is granted privileges to write to protected folders. However, whitelisted applications themselves are not protected from being misused by other applications. This trust is therefore unjustified, since a malware can perform operations on protected folders by using whitelisted applications as intermediaries. In particular, ransomware might be able to exploit some of the whitelisted applications to change the contents of files on their behalf, thus to encrypt user data. Similarly, personal files of users might be irrevocably destroyed by a wipeware.

In this chapter we play one move more of the game. We suggest a new principle that, if followed and properly implemented, render AVs' and OSs' resilient to the attacks (*Cut-and-Mouse* and *Ghost Control*) that we have found.

5.1 Background

In this section, we recap the essential background information to understand our attacks. We begin with explaining the ransomware mitigation in current antivirus solutions. Next, we summarize existing measures provided by Windows OS to protect processes from unauthorized modifications.

Ransomware Defense in AVs

In response to the rise of ransomware threat, AV vendors have developed dedicated ransomware detection modules that are either integrated into their products or as standalone tools. While internal mechanisms of AVs are not publicly documented, the available options in most of AV configuration interfaces suggest that these anti-ransomware components are primarily based on whitelists. Similar to the virus signature databases, these lists are maintained by AV vendors by default, though, users can also add additional applications that they trust.

The vendor of Windows OS, Microsoft, has also developed a specific anti-ransomware solution, called *Controlled Folder Access*, which has been included in Windows 10 Fall Creators Update (Version 1709) and Windows Server 2019. Ransomware Protection, integrated into Windows Defender antivirus, controls which applications have access to protected folders, a list of directories that includes system folders and default directories such as `Documents` and `Pictures`. Users can also add further directories to the protected folder list in order to extend the coverage of protection. By default, the decision of granting applications access to protected folder is made by Windows, hence Microsoft, but users can also allow specific applications to access the protected folders.

Throughout this chapter, we use the term *trusted applications* when referring to the applications that has write access to protected folders, either granted by AV vendor or added by the user.

Process Protection via Integrity Levels

Computer architecture we use today is designed to run multiple processes concurrently, that is, all running processes share the same execution environment. To protect processes from malicious alterations by other processes, Windows OS employs access control mechanisms. Mandatory Integrity Control (MIC) is one of these security features, which enables the OS to assign an Integrity Level (IL) to a process: this value indicates the privilege level of that process. MIC defines four values for IL, with the increasing privileges: *Low*, *Medium*, *High*, and *System*. When a process attempts to interact with another process, MIC checks IL of the initiator and prevents if the target has higher IL. For example, injecting code to another process using `CreateRemoteThread` or write data to the memory of another process via `WriteProcessMemory` will fail if the caller does not possess at least the same IL as the target.

Closely related to MIC, User Interface Privilege Isolation (UIPI) is another security feature of Windows, which complements MIC to prevent unauthorized process interactions. UIPI also utilizes IL and blocks window messages flowing from a process with lower IL. For example, calls to `SendMessage` API would fail if the caller has a lower IL than the target. Specifically, UIPI prevents the Shatter attack that we review in §5.7.

5.2 Threat Model

In the description of our attacks, we assume the system is protected using the latest generation of AVs with specific modules against ransomware, and with built-in anti-ransomware feature of the OS. We assume the attacker is able to get access to a Windows system with user privilege levels by either tricking the user into clicking on a file (e.g., attached or linked in an email) or by exploiting a vulnerability in the victim's system. Once the attacker has established a foothold into the system, it will typically drop/download a malware to perform malicious operations, however, the malware will be blocked by an AV, or in the case of ransomware, encryption of files in protected folders will be blocked by anti-ransomware protected folder feature offered by Windows or some AVs. Henceforth, the focus of this chapter is on how attackers can bypass AVs and anti-ransomware protection modules, and in providing practical mitigation solutions, rather than in the problem of detecting and protecting the system from remote attacks. This threat model is sometimes referred as a *2nd stage attack*, meaning an attacker would need to have remote access to a victim's computer, or have installed a malicious application using one of the two previously outlined alternatives (or through other means).

In this threat model, we will perform two attacks, which are described in the next two sections: the first attack (*Cut-and-Mouse*) is aimed at bypassing the protected folder feature to encrypt files in protected folder, while the second one (*Ghost Control*) is aimed at disabling AVs' real time protection.

5.3 Cut-and-Mouse: Encrypting Protected Folders

In this section, we describe our attack, *Cut-and-Mouse*, which allows ransomware to evade detection by anti-ransomware solutions, which are based on protected folders, and to encrypt the victim's files. First, we investigate the root causes that leads to this attack. Next, we give the attack details, and finally propose a practical solution.

Disharmony Between UIPI and AVs

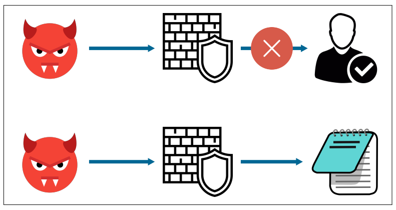
As explained in §5.1, anti-ransomware modules of commercial AV software grant write access to trusted applications only. To ensure this defense strategy cannot be easily bypassed, the trusted applications should be protected from any malicious modifications which would be seen in a typical malware attack. For instance, as we report the details in §5.5, current AVs detect when a malicious DLL module is injected into a trusted application, and suspend or kill its process. Similarly, UIPI, another protection described in §5.1, protects processes that run with administrative privileges from malware.

Nonetheless, we have discovered two entry points for an attack which enables malware to bypass these defense systems, namely:

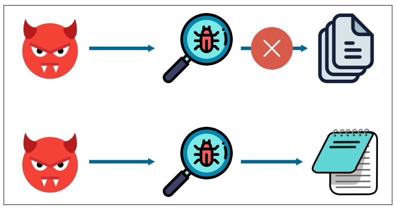
(**E**-i) *UIPI is Unaware of Trusted Applications*: UIPI filters simulated inputs based on integrity levels, however, UIPI is agnostic of the trust level assigned to applications, so it does not enforce any policy in these cases: as shown in Fig. 5.1a, that means that an attacker can send messages to trusted applications, in particular to those that are allowed to read and write to protected folders;

(**E**-ii) *AVs Do Not Monitor Some Process Messages*: AVs do not monitor synthesized clicks or key press events flowing into the trusted applications: as depicted in Fig. 5.1b, this means that a ransomware can bypass protected folder enforcement by sending control messages to a trusted application.

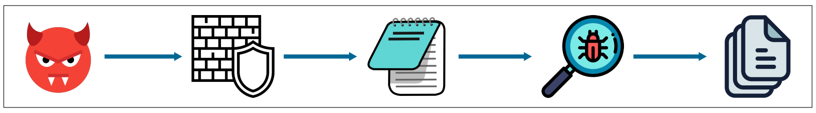
These two entry points form a vulnerability that can enable malware to perform practical attacks, such as that shown in Fig. 5.1c where a ransomware can control a trusted application to perform controlled write operations as to encrypt inaccessible protected files. The attack is described in more detail in the next section.



(a) Ransomware's messages to high IL applications are blocked by UIPI (*top*); but ransomware can send messages to trusted applications (*bottom*).



(b) Ransomware's write attempts to protected files are blocked by AVs (*top*); however, sending messages to trusted applications is allowed (*bottom*).



(c) Ransomware can control a trusted application to perform write operations to protected files.

Figure 5.1: The disharmony between UIPI and AV software's protected folders mechanism, as described in (a) and (b), is the root cause of the vulnerability which leads to the attack depicted in (c).

Attack Overview

Using the vulnerability described in the previous section, ransomware can bypass anti-ransomware protection via controlling a trusted application and encrypt the files of the victim, including those stored in protected folders. To this end, for each file F_{target}, the ransomware performs the following tasks as depicted in Fig. 5.2. Firstly, ransomware reads the contents of F_{target}, which is in a protected folder (1). This is perfectly legal: in fact, reading a protected file is permitted by default[2]. The plaintext retrieved from F_{target} is encrypted in ransomware's own memory. The resulting ciphertext is then encoded in a suitable encoding format, e.g., Base64 [92], and copied into the system clipboard (2). Next, the ransomware launches the `Run` window (3) to start a trusted application $App_{trusted}$, with the goal of controlling it. In this example, $App_{trusted}$ is `Notepad` as it is typically trusted in Windows environments. In addition, `Notepad` understands shortcuts for file and edit commands that ransomware will send. Using the `Run` window, ransomware executes $App_{trusted}$ with the argument F_{target}, so that the contents of F_{target} is loaded into $App_{trusted}$'s window (4). Next, the data in $App_{trusted}$'s window are selected, and overwritten with the clipboard data (the encrypted data) with a paste command (5). Finally, $App_{trusted}$ is instructed by the ransomware to save the modifications, and close the handle to F_{target} (6). All interactions in Steps 3-6 are carried out by sending keyboard inputs which are synthesized programmatically by the ransomware to control $App_{trusted}$.

2: Some AVs also provide an optional, more strict access setting that, if activated, makes AVs block the read requests from non-trusted applications.

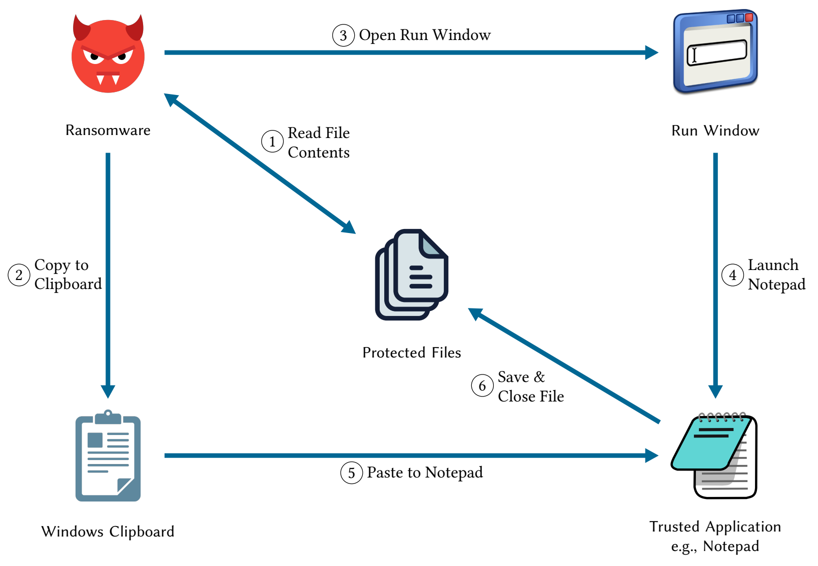


Figure 5.2: Bypassing anti-ransomware protection of AVs by using inputs programmatically synthesized by ransomware to control a trusted application.

Algorithm 6 *Cut-and-Mouse* Attack: Exploit Trusted Apps with Simulated Keyboard and Mouse Inputs to Write to Protected Folders.

```
1:  procedure CUT-AND-MOUSE($App_{trusted}$)
2:      $FileList$ ← EnumerateTargetFiles()
3:      for all $f \in FileList$ do
4:          $plainBytes$ ← $f$.ReadAllBytes()
5:          $encBytes$ ← Encrypt($plainBytes$)
6:          $encodedText$ ← Base64($encBytes$)
7:          CopyToClipboard($encodedText$)
8:          Simulate(Run, $App_{trusted}$ <$f$>)          ▷ Win+R
9:          Simulate(SelectAll)                        ▷ Ctrl+A
10:         Simulate(Paste)                            ▷ Ctrl+V
11:         Simulate(Save)                             ▷ Ctrl+S
12:         Simulate(Close)                            ▷ Alt+F4
```

The combination of these actions effectively allows ransomware to bypass the current protection methods of AVs that are aimed explicitly at blocking ransomware. Therefore, by referring to the never-ending 'cat-and-mouse' game of detection/anti-detection and anti-evasion/evasion among AVs and malware, and the usage of simulated keyboard and mouse inputs, we have named this attack *Cut-and-Mouse*. Algorithm 6 details the main steps of the *Cut-and-Mouse* attack.

In more detail, there are two steps that are required for the *Cut-and-Mouse* attack to be successful. First, the step *Open Run Window* (3) in Fig. 5.2 is needed to disguise the operation of starting a trusted application as if it was executed on behalf of the user. If, instead, `Notepad` is directly executed by the ransomware, AVs would block write requests even if the rest of the attack is performed as described previously. In fact, in this example, even if `Notepad` is a trusted application (therefore allowed to write on protected folders), its parent process would be the ransomware, which is not trusted by the AVs, hence, write operations would be blocked. Secondly, as noted in Sidenote 2, the step *Read File Contents* depicted in (1) in Fig. 5.2 can be blocked by AVs in some circumstances. For this reason, this limitation (that of not being able to read file contents) can be circumvented if ransomware exploits a trusted application to access the content on behalf of the ransomware. For example, ransomware could instruct `Notepad` to open the target file, and then synthesize two keyboard press events for `Ctrl+A` (Select All) and `Ctrl+C` (Copy), which would allow the ransomware to select all the content of the file and copy it to the system clipboard. Since the clipboard is shared between all running processes, ransomware can easily obtain the clipboard contents. It should be noted that, though, this technique might result in unrecoverable data loss with binary encoded files, due to the the presence of non-printable characters displayed by `Notepad`. However, ransomware can detect the content of the file before deciding which file to encrypt.

Proposed Mitigation Strategy

As a simple yet effective countermeasure to protect AVs modules against our *Cut-and-Mouse* attack, we suggest that trusted applications should not receive messages from non-trusted applications. That is, AVs must intercept all messages flowing to a trusted process and block (or discard) the messages sent by non-trusted processes. This countermeasure is analogous to what UIPI already implements to guarantee process privileges. It should be noted that, however, UIPI is not provided with a whitelist of AVs: therefore, it cannot enforce such a filtering in practice and this defense task should be fulfilled by the AV programs.

We elaborate more on this strategy in Section 5.8, where we define a requirement that a secure message filtering system should at least have.

5.4 Ghost Control: Disabling Antivirus Software

In this section, we describe how the simulation attack *Cut-and-Mouse* described in §5.3 can also be effectively applied in other scenarios, and we also attempt to hypothesize how it can be used in future attacks.

In the course of our analysis, we have found a surprisingly simple utilization of synthesized mouse events technique, which would allow an attacker to deactivate nearly half of the consumer AV programs, including some popular products. We start by explaining the reasons for the presence of deactivation functionalities in AVs. Next, we describe the steps to perform the attack (*Ghost Control*), investigate the weakness in detail, and propose a practical solution to fix it.

Necessity of the AV Deactivation Function

Signature-based detection has been the primary defense method of AVs, and naturally, this technique is efficient only against known malware as it can be bypassed easily, e.g. by obfuscation/packing and polymorphic malware. To minimize this limitation, nearly all current AVs employ some heuristics to detect malware by monitoring behaviors of processes and looking for anomalies. However, this functionality comes with a price: occurrences of *false positives*. In the context of malware defense, false positive is the situation where an AV software flags a benign executable as malware, and it usually proceeds with termination of the associated process, hence interrupting the user. For example, when a user installs a new software package, the installer may write to system directories, modify the Windows Registry and configure itself to run when the user logs in. The behavioral decision engine of an AV may be confused by these activities, which indeed might look suspicious as they are largely used by malware. Therefore, an AV may prevent the software from being installed correctly. Consequently, some vendors recommend the users to turn off their AV temporarily for a successful installation of their benign application, for instance [93]. Moreover, some special software may require AV to be disabled while running, for instance [94]. As a result, AV companies provide users with a switch that can be used to deactivate the real-time protection for different periods of time, ranging from a short period, such as 2 minutes, to longer periods, such as 2 hours, or until the computer reboots. Of course, the ability to "freeze" an AV might lure attackers to abuse this functionality to bypass malware detection, hence, AVs should offer ways to ensure that this functionality can be disabled only by authorized users.

Stopping Real-time Protection

In our second attack, *Ghost Control*, we show how an attacker can disable the AV protection by simulating legitimate user actions to activate the Graphical User Interface (GUI) of the AV program, and then to "click" the turn-off button. The proposed attack comprises two phases. The first phase is performed off-line by the malware author. In this phase, the developer collects the required pieces of information about the user events to be simulated to successfully disable the AV. This set of information consists of (i) x and y coordinates on the screen; (ii) which mouse button to be simulated; and (iii) duration to wait until the next menu is available. Please note that the mouse coordinates should lie in the correct area on the screen for this attack to work. In addition, these values would change from victim to victim, or even in the same host, as the screen dimensions vary or would differ under various resolutions. Therefore, the malware author needs to collect the correct locations of the menus of all the major AVs under different display settings to increase the effectiveness. For example, this would require the attacker to install the target AVs in virtual environments with different screen dimensions to collect the necessary data. Once data collection is completed, malware author embeds that information into the malware executable to be used during the attack (alternatively, malware can download the required information from a remote server at the time of attack).

The second phase of the attack is the actual malicious step which starts immediately after the infection. On the victim machine, malware performs a reconnaissance work to determine the installed AV product(s) and obtain the screen dimensions. Next, malware prepares the event sequence to be simulated to turn off the AVs, and synthesizes the keyboard and mouse events accordingly. Algorithm 7 illustrates the part of *Ghost Control* that is responsible for the turning off of the installed AV program.

Algorithm 7 *Ghost Control* Attack: Disable Real-Time Protection of AV with Simulated Events.

```
1: global EventSequenceDatabase as EvSecDB
2: function TurnOffProtection
3:     antivirus ← GetInstalledAV()                  ▷ AV to deactivate.
4:     events ← EvSecDB.GetEventSequenceFor(antivirus)
5:     for all $e \in$ events do
6:         Simulate($e$)
7:     return Success
```

As a consequence, the range of functionalities that *Ghost Control* enables to malware authors is very large, some having a high impact: for instance, once the real-time scanning is stopped, malware can be instructed to use *Ghost Control* to drop and execute any malicious program from its C&C server.

Proposed Mitigations

In order to develop a robust defense against this vulnerability, we need to understand the root causes behind this vulnerability. Our analysis shows that there are two reasons why *Ghost Control* is able to deactivate the shields of several AV programs:

(**W**-i) *AV Interface with Medium IL.* Processes related to the AV main interfaces that manage these defense systems run in such a way that they are accessible from processes that run without administrative privileges. It is therefore possible to send “messages” from any process to these process, e.g., mouse click events, without any restriction.

(**W**-ii) *Unrestricted Access to Scan Component.* The scanning components of vulnerable AVs do not require the user to have administrative rights to communicate to them, e.g., they can receive a `TURN_OFF` message from any process. Consequently, *Ghost Control* can initiate and control the reaction which involves accessing this critical component of AVs.

(**W**-ii) is actually a more critical vulnerability than (**W**-i). In fact, if an AV software has (**W**-ii), then malware can skip interacting with the GUI of AVs through (**W**-i) to directly communicate with the AV’s scanner component and send a `TURN_OFF` message. This is in fact only a practical limitation: for instance, in our experiments (see Section 5.5), we have noticed that AV12 employs CAPTCHA mechanisms to verify that the user really wants to turn-off the protection. Even if we assume the CAPTCHA is a solid measure against automated attacks[3], however, malware can still bypass the CAPTCHA verification by directly accessing the scanner component due to (**W**-ii).

3: We note that CAPTCHA can actually be bypassed using other means, e.g. with CAPTCHA solving services, but they might not always be applicable.

To mitigate the root causes of the failure of the affected AVs, we propose a solution based on the following principles:

(**F**-i) *Elevated AV Interface.* AVs should run the main GUI interface with administrative privileges. By doing so, AV processes will have high IL, and AVs would not receive the messages of *Ghost Control* or any other malware since UIPI would drop the unauthorized messages.

(**F**-ii) *Restricted Access to Scan Component.* AVs should design and develop their scan components in such a way that accessing it would require the user to have administrative rights.

Mitigation at OS Level Windows platform provides a tool to monitor critical components and applications, including virus protection, firewall, and OS updates. This tool, called Windows Security Center (WSC), reports the security status of the system to Action Center. For example, WSC detects if an AV program is installed, and continuously checks if the AV is turned on and up-to-date. If, for some reason, real-time protection of the AV stops, WSC informs Action Center, which notifies the user and provides an interface to take a remediation action.

We propose to adapt this architecture to detect and prevent (**W**-i) and (**W**-ii) as follows. AV programs can already be integrated to Action Center by registering themselves with WSC. During registration, the path of the main executable of the AV program is supplied to WSC along with the product name and other pieces of information. WSC can use these data to perform security checks on the AV executable, in particular, WSC can

1. prevent the registration of the AV if the AV's interface is configured to run with medium IL;
2. auto-escalate the integrity of the AV process to high; or
3. set the security descriptors of a AV components to default value so that accessing them requires system or administrator privileges.

The first option, preventing installation of the AV, can be viewed as undesirable, especially considering the availability of the second and third options. However, it should be noted that a mitigation that include raising an error would ultimately allow the developers to be aware of the vulnerability, and might lead them to discovering other issues that would remain hidden otherwise.

In the next section, we discuss and share the results of our experiments, which show that (i) some AVs are vulnerable to *Ghost Control* (ii) the proposed measures are actually employed by some AVs that, therefore, are not vulnerable to the *Ghost Control* attack. From that evidence, we conclude that these attacks are able to circumvent several off-the-shelf AVs; and the proposed mitigation is both effective and practical to use in real-world systems.

5.5 Experimental Results

To demonstrate the impact of the exploitation of the vulnerabilities described in Section 5.3 and Section 5.4, we developed three proof-of-concept prototypes for the attacks, and tested them against consumer products of 29 AV companies. In this section, we detail the dataset and test environment of our experiments, and report our findings.

Dataset and Test Environment

The list of the AV programs that we would test in our experiments was determined from the reports of independent organizations that test AV products. We populated our initial list with the AVs from the recently published reports of AV-TEST [95] and AV-Comparatives [96]. The initial list had AVs from 35 vendors, however, some vendors discontinued their consumer AV product, or were not available for download. In the end, our dataset contained 29 AV programs from world wide vendors.

We conducted all experiments on a VM running Windows 10 Pro x64 Version 1903 (OS Build 18362.30) OS. After a fresh installation of Windows 10, we updated the system and created a snapshot of a template VM. Next, in each run of the experiment, we restored the VM to the snapshot and installed the latest version of the AV software to be tested (available at the time of this writing), which was usually determined by the installer application downloaded from the vendor's website. Finally, we updated the database of the AV software to obtain the latest signature definitions and heuristics.

Attacks Detected by AVs

We first verified whether AVs are able to detect and block known attacks aimed at bypassing the anti-ransomware module. In the first experiment, we injected a malicious DLL into a trusted application, where the DLL would start encrypting the default files protected by AVs. As expected, all of the 29 AVs in our dataset detected this technique, and suspended (or sometimes killed, e.g., `AV17`) the injected trusted application before the first write operation, as DLL injection is one of the oldest attack techniques.

The next experiment was aimed at maliciously controlling a trusted application to save encrypted content to protected files. In this attack, we instructed a ransomware program implemented in C# language to launch the trusted application using `Process.Start` method. As expected, this attack is also not effective as the trusted application is created as a child process of the ransomware, which is not trusted, and therefore blocked by AVs.

Lastly, we executed a ransomware with elevated privileges while protected folders feature of AVs were active. The sample, instead of using our *Cut-and-Mouse* technique, is designed to directly encrypt and overwrite the files in `Documents` and `Pictures` folders. Again, all AVs in our dataset detected the attack and blocked the malicious operations, which shows that protected folders feature of AVs is immune to ransomware having admin privileges.

Encrypting Files in Protected Folders via Simulated Inputs

In this section, we report the test results where attack is run against AVs. First, we describe the technical requirements for the successful exploitation of attack, and our implementation.

Technical Requirements

Successfully performing attack requires a trusted application that should be available on the victim's machine. Furthermore, this specific trusted application should possess the capabilities to: (i) be started from command line; (ii) accept file paths as argument; (iii) edit/manipulate files; and (iv) receive inputs from clipboard. We have discovered that the best candidate that fulfills all these requirements is the `Notepad` application, since it is one of the most commonly-used built-in Windows application, and it is digitally signed[4], therefore, whitelisted by AV programs. In addition, file size limit of `Notepad` is 56 MB on Windows 7, while it can open documents with size more than 512 MB on Windows 8.1. To send data to from a ransomware sample to `Notepad` application, we exploit Windows Clipboard, which stores objects that can be shared between all running applications. The memory area to store these objects are allocated using `GlobalAlloc` function. On 32-bit systems, virtual memory of a process is limited with 2GB, which also determines the maximum capacity of the clipboard. This gives us a sufficiently large memory space to store encrypted and encoded data, so makes the clipboard suitable to use as a swap area in our attack.

4: The digital signature of Notepad, as is the case for many built-in Windows applications, is not embedded in the binary but can be found in the appropriate catalog file.

Implementation

We implemented a prototype of in C# language, using .NET Framework version 4.6.1. The prototype synthesizes only keystrokes as input simulation, for which, `SendInput` is employed.

Our prototype implements Algorithm 6 and works as follows. First, all of the files in the target directory are enumerated using `Directory.GetFiles`, and the files with the target extensions are filtered. Namely, in the experiments, we targeted the following file extensions: `.docx`, `.xlsx` and `.png`. Next, using `Clipboard.SetText`, ransomware copies the command `attrib.exe -r targetPath\*.*` to the clipboard, where `targetPath` is replaced with the absolute path of the target directory. We instructed the ransomware program to simulate keystrokes `Win+R` to open the `Run` window, and `Ctrl+V` and `ENTER` to run the copied command. This step ensures that the read-only attribute was removed from the target files.

Next, for each file, our prototype proceeds as follows. Firstly, the file is read as binary using `File.ReadAllBytes` and then, using `AesCryptoServiceProvider`, the content of the file is encrypted in memory. After this, the byte stream is converted into printable text using Base64 encoding, and copied to the system clipboard. As previously discussed, our prototype uses `Notepad` as $App_{trusted}$, so it executes `Win+R` command, sleeps 500ms while waiting for the `Run` window to open, and then pastes the command `notepad.exe targetFile` into the `Run` window, where `targetFile` is replaced with the absolute path of F_{target}. At this step, the prototype sleeps for an additional 500ms to ensure that `Notepad` window is opened – this window displays the contents of the file. Next, the prototype sends the keystrokes `Ctrl+A` to select all the text in the `Notepad` window and `Ctrl+V` to paste the clipboard data into it, which replaces the selected content with the ciphertext. Here, the prototype performs one final sleep of 500ms to ensure that all the data are correctly pasted into `Notepad`. To save the file, `Ctrl+S` command is sent to `Notepad`, which effectively overwrites the file with the encrypted data. Finally, `Alt+F4` command is sent to close `Notepad`.

Test Results of Cut-and-Mouse Attack

After installing the AV software on the VM snapshot, we placed decoy files in the `Documents` and `Pictures` folders of the user – these are both protected folders, hence protected from ransomware attacks. Next, we run our *Cut-and-Mouse* prototype and checked the effect of the attack on the files.

At the end of each run, the decoy files were overwritten with the pasted data successfully. The results demonstrate the effectiveness of the *Cut-and-Mouse* attack, which was able to bypass all 29 AV programs in our test set and encrypt the files in the protected folders. To the best of our belief, *Cut-and-Mouse* is a new attack that controls legitimate applications for malicious purposes via simulated user inputs. The evidence that even the latest AV products cannot detect this attack suggests that this new attack type can cause more damages if used by real-world attackers with different –and possibly creative– ideas to perform powerful exploitation of systems.

Destructive Cut-and-Mouse: Wiping Files in Protected Folders

Although *Cut-and-Mouse* attack is effective on AVs, the limitations of using `Notepad` forms a performance barrier when the file size noticeably increases. To remove this bottleneck, we will use another built-in Windows application, `Paint`, as intermediary.

`Paint` also satisfies all the technical requirements in Section 5.5 with a couple of exceptions. First, only some image files are accepted as a file argument. `Paint` raises an error when the user tries to open a `.PDF` document, for example. Secondly, `Paint` only accepts a valid image from clipboard. If the image in clipboard is corrupted, it cannot be pasted to `Paint`.

The first limitation can be overridden easily by adding a file extension explicitly, which would allow `Paint` to write to any files. The second limitation, however, makes it difficult to build an “honest” (i.e., fully working) ransomware, as it requires the implementation of a reversible encoding technique to transform arbitrary data to a valid image format. Instead, for the scope of this research, we demonstrate another type of malware, known as *wipeware*, able to overwrite user’s files with a randomly generated image to destroy them permanently.

As in Section 5.5, *Cut-and-Mouse* wipeware also starts with collecting target files and removing their read-only attributes. Next, the wipeware prototype creates a random bitmap image which has the same size of the largest file. The random image is copied to clipboard by calling `Clipboard.SetImage`. Then, for each target file, the wipeware performs the following tasks in order: (i) synthesize `Win+R`, programmatically type `mspaint.exe` in the `Run` window, and synthesize `ENTER` to open `Paint`; (ii) synthesize `Ctrl+V` to paste the random image from clipboard after

`Paint` windows appears; (iii) synthesize `Ctrl+S` to save the file, which would open up the `File Save` dialog; (iv) programmatically type the full path of the file and synthesize `ENTER`; (v) synthesize `ENTER` to confirm the overwrite message box; (vi) synthesize `Alt+F4` to close `Paint`. By sleeping 500ms between each step to ensure all operations are carried out correctly, our *Cut-and-Mouse* wipeware could destroy each decoy file in a few seconds.

Controlling Real-Time Protection of AVs

In order to demonstrate the feasibility of our attack in Section 5.4, we implemented the prototype of *Ghost Control* in C# language, using .NET Framework version 4.6.1. To collect the coordinates of the mouse on the screen, the prototype uses `GetCursorPos()` API. For synthesizing keystrokes, mouse motions, and button clicks, `SendInput()` API is used. Between each simulated mouse clicks, the prototype sleeps for 500ms to ensure that the next menu on the GUI is available to be selected.

Collecting Coordinates to Disable AVs

After installing the target AV, we performed cursor movements towards the tray icon area as to select and click the AV icon[5] and used AV's GUI to disable the real-time scanning using the provided menus. During this procedure, we recorded the (x, y) coordinates of the cursor and the types of clicks that we had performed until the protection was disabled, i.e., AV's security notification appeared. For instance, Figure 5.3 shows the console output of the application we used to collect mouse coordinates while a real user disables `AV27` on a VM with screen resolution set to 1920x1080. For the duration of the deactivation, we used the default values suggested

5: For the sake of proof-of-concept, we did not implement a function to detect AV's icon among the tray icons. Actual malware would need to do that, for example, by checking window titles to find AV's icon, but this is not a difficult routine.

```
Left Click   x=1868, y=992  // Show Tray Icons
Right Click  x=1866, y=952  // Open AV's Menu
Move Cursor  x=1860, y=873  // Change Settings Submenu
Left Click   x=1700, y=877  // Real-Time Scan Settings
Left Click   x=1315, y=430  // Turn-off Button
Left Click   x=1280, y=555  // Verify Turn-off
```

Figure 5.3: Console output of the application which sniffed the real user actions while disabling `AV27`.

by AVs to freeze their functions. The minimum length is usually set to be 15 minutes, which is a sufficient time frame to successfully conduct an effective attack. Here, the attackers could also select an option that gives them a longer time-period.

Stopping Real-Time Protection

Using the collected coordinates of the AV's menus and buttons, we instrumented the recorded actions and parameters into our *Ghost Control* prototype, which is used to exploit the specific AV that we tested in each experiment. Next, we run the *Ghost Control* prototype and waited until all the events are simulated.

If *Ghost Control* attack succeeds, a warning window appears which notifies the user that the computer is not protected. In some experiments, we even went further and simulated mouse clicks to remove this notification window, which would be expected from a real-world malware. This shows how this class of attacks can be further extended to perform potentially more powerful malicious actions.

Product	IL of GUI	Utilizes UAC	Vulnerable to *Ghost Control*
AV1	Medium	✓	
AV2	Medium	✓	
AV3	Medium	✓	
AV4	Medium		✓
AV5	Medium		✓
AV6	Medium		✓
AV7	Medium		✓
AV8	Medium		✓
AV9	Medium	✓	
AV10	Medium	✓	
AV11	Medium Plus		
AV12	Medium		✓
AV13	Medium	✓	
AV14	Medium		✓
AV15	Medium	✓	
AV16	Medium		✓
AV17	Medium	✓	
AV18	High		
AV19	Medium	✓	
AV20	Medium		✓
AV21	Medium	✓	
AV22	High		
AV23	High		
AV24	Medium		✓
AV25	High		
AV26	Medium		✓
AV27	Medium		✓
AV28	Medium		✓
AV29	Medium		✓
Tested: 29			Vulnerable: 14

Table 5.1: Evaluation of AV products. Check marks under *Vulnerable to Ghost Control* denotes that the AV product was successfully disabled by *Ghost Control*.

As shown in Table 5.1, during our experiments on 29 AV products, we detected that 14 AVs could be efficiently deactivated by *Ghost Control* using our attack in Section 5.4. According to a recent report by OPSWAT [97], the market share of AVs that are vulnerable to *Ghost Control* is more than 29%[6]. Furthermore, 6 of these AVs have been frequently rated as "TOP PRODUCT" in the reports of AV-TEST, and 3 of them received 3-stars (best rating) from AV-Comparatives. It is surprising for us that such a critical vulnerability, arguably one of the worst that an AV might have, is found in such a large share of AVs.

6: We were not able to calculate the exact statistics as the shares of the 10 AVs that we could stop are consolidated into "Other".

In the experiments in which *Ghost Control* was not able to successfully disable the AV, we noticed that this was due to two factors. First and foremost, User Account Control (UAC) prompt which uses MIC stopped *Ghost Control* attack. In these cases, after *Ghost Control* generated a click event to turn-off protection, UAC notification appeared, which always runs with high IL. However, since *Ghost Control* is a medium IL process, it was not be able to bypass UAC verification successfully. Secondly, as shown in Table 5.1, 5 AVs always run with medium plus IL or high IL. Consequently, UIPI filters and drops the events that our *Ghost Control* prototype synthesize and sends to these AVs.

5.6 Security Analysis of Auxiliary Measures

During the experiments, we were confronted with two additional security measures, namely sandboxing and CAPTCHA verification, which protected AVs from *Ghost Control*, even if the GUI of the tested AV was vulnerable. In this section, we will look at these measures and we explain how we were able to bypass them.

Insecure Sandboxing Methods

In the security context, sandboxing is a mechanism to run an unknown application in a controlled environment, isolated from the host. The main purpose of employing sandbox in AVs is to prevent previously unseen malware from damaging the system which would evade the signature-based detection otherwise. Although the high level understanding of sandboxing is common to all AVs, the implementation details might vary between different vendors. In addition, AVs do not publicly share the inner workings of their sandboxes, so we can only guess the capabilities of sandboxes from their whitepapers and AV settings.

Most vendors in the AV industry supply sandbox products for their business-level customers, usually as a gateway device to be integrated into the network. Some AVs provide cloud-based sandboxes for home users where unknown files are submitted for analysis. For example, AV6 offers its users a cloud service to analyze files with certain extensions. With that said, we could identify that only AV1, AV2, and AV7 let users run programs in a virtual, isolated environment on their computers. Other AVs might also have a built-in sandbox technology, but according to our observations, they do not expose any settings, show any notifications indicating a sandbox, nor advertise any information among the products' features.

According to our experiments, both AV1 and AV2 execute each unknown program in a sandbox at the first run. This automatic execution seems to be time limited, and takes around 30 seconds. After that, the programs are automatically started in the host environment without the sandbox restrictions. Users can also run programs in the sandbox without a time limit using the context menu. From program outputs and error messages, we infer that both AVs create a virtual file system where the programs being tested cannot access the files on host, even for reading. The programs can access to the Internet though, and the trusted files they download can be saved outside the sandbox. Furthermore, programmatically synthesized events, such as simulated mouse clicks and key strokes cannot reach outside the sandbox. By sleeping for 30 seconds in total and meanwhile performing a benign task like printing to console, our *Ghost Control* prototype was found harmless by the sandbox of AV1 and AV2, however, it was stopped by UIPI.

Differently, AV7 does not apply a time limit for the automatic sandboxing at the first run. Moreover, it allows the programs being tested to read the actual files on the host. From the company website and program interface, we infer that the sandbox of AV7 prevents any running application from writing to any file or registry by placing function hooks, sending inter-process messages and window messages, or synthesizing keyboard events.

The enforcement is applied even to the processes that run with admin privileges, therefore, an unknown process' device driver installation is also denied by the sandbox.

However, we noticed that the sandbox of AV7 is developed in such a way that mouse clicks are not filtered, and therefore a malware can easily escape the sandbox. In other words, it could synthesize mouse clicks using `SendInput`. During our experiments, even if all unknown applications were automatically run in the sandbox, mouse events synthesized by our prototype were not filtered by the sandbox. Combined with the vulnerable GUI of AV7, our commands received by AV7 and we could stop the real-time protection. Our conclusion from this observation is to add a fix to the weak sandbox by including mouse events to the filtered operations, in addition to the fixing the vulnerable GUI of AV7.

Passing Human Verification

CAPTCHA (Completely Automated Public Turing Test To Tell Computers and Humans Apart) is a challenge-response test to determine if the user is a human or not [98]. CAPTCHA is a widely adopted technology on the Internet to distinguish humans from computers. In our experiments with 29 AVs, we identified that two AV programs, AV10 and AV29, utilized CAPTCHA images in their program flow, as illustrated in Figure 5.4 and Figure 5.5.

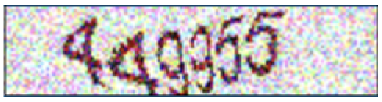


Figure 5.4: AV10 generates CAPTCHA codes that contain only numeric characters.

AV10 shows a CAPTCHA image and asks the code therein when Shut down protection (which terminates the main AV process and stops protection) is clicked to ensure that the GUI interaction is engaged by a human, not a malware. The vendor of AV10 indicates that this measure is employed to prevent automated shutdown by malware. However, no verification is performed when Disable protection is clicked—in which case, the AV program continues to run, but protection service is stopped. As a result, *Ghost Control* could easily disable AV10.

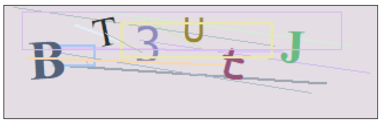


Figure 5.5: AV29 generates CAPTCHA codes that consist of lowercase and uppercase letters, and numbers.

When testing our attack to disable AV29, during the final step of the turn-off sequence, this AV generates a CAPTCHA image and displays on the screen to verify that the request comes from a genuine user. The user must enter the code correctly for AV protection to be turned off. In our first try, our *Ghost Control* prototype failed to turn off AV29 as it could not enter the CAPTCHA code. To overcome this limitation, we enhanced the prototype with the ability to partially capture the screen which contains the CAPTCHA code. Next, the prototype sends the captured image to an external user, who can solve the CAPTCHA and sends the correct CAPTCHA code back to our prototype, which synthesizes the code to AV29, and completes the turn off sequence.

Although our method for solving CAPTCHA codes might look like a naïve and impractical solution that doesn't scale, please note that cybercriminals have at least two alternatives, as follows.

- Capture the CAPTCHA image and dispatch to C&C server where a CAPTCHA-solver program is running, and return the code to the malware. The latest advancements in Machine Learning techniques allow to develop highly accurate text-CAPTCHA solvers [99]. The

CAPTCHA images shown in Figure 5.4 and Figure 5.5 might be solved by an automated software.
- Use CAPTCHA-solver services available to solve CAPTCHAs for affordable prices with high success rates to make the attack scalable and profitable [100].

5.7 Related Attacks in the Literature

In this section, first we review existing attacks involving simulated inputs to perform malicious actions. Next, we outline previous research on the security of antivirus software.

Attacks Related to Input Simulation

Input simulation is the practice of programmatically synthesizing input events, such as mouse clicks or key strokes, which are typically performed by the user. This section describes some the most powerful existing attack techniques that make use of input simulation.

Ghost Clicks

In [101], Springall *et al.* developed a proof-of-concept malware to manipulate votes in Estonian Internet Voting system. On infected clients, the malware simulates keyboard inputs to activate the electronic identifier (e-ID) of voters and submit a vote in a hidden session that is invisible to the voters.

Recently, under a different threat model, in [102] Maruyama *et al.* demonstrate a method to generate tap events on touch screens of smart phones using electromagnetic waves. In this scenario, the victim's device can be forced to pair with a malicious Bluetooth device once it gets in the range of the attackers. Even if the victim denies the pairing by choosing `CANCEL` in the security prompt, the attacker can alter this selection and make the OS to recognize user input as `CONNECT`.

Pay-per-click advertising systems are also vulnerable to fake clicks, which is known as Click Fraud [103]. In these systems, the advertisers get paid according to the number of clicks on advertisements. By generating fraudulent clicks on the ads, a malicious advertiser can increase its payment.

Perhaps the attack closest to the one described in this chapter is *Synthetic Clicks* [104], credited to Patric Wardle [105]. Exploiting a bug in macOS OS, the attacker could send programmatically-created mouse clicks events to security prompts that would result in vertical privilege escalation. This way the attacker could cause any damage, including retrieving all of the user's passwords stored in the keychain. Our attacks, *Cut-and-Mouse* and *Ghost Control*, target AVs, not OS, do not rely upon a bug in the OS, and can be used to instruct a trusted application to perform different malicious operations.

Reprogramming USB Firmware

In [106], Nohl *et al.* demonstrated that it is feasible to modify the firmware of a USB device, for instance a USB stick, to behave like a keyboard. Known as BadUSB, this technique works by reprogramming the device's firmware in order to type commands on the victim's computer. When plugged into a computer, the malicious USB device can simulate the key strokes of the user, for example, type and execute a script which downloads and runs a malware.

Shatter Attack

In [107], Paget describes a weakness in Windows OS that allow a process to inject arbitrary code into another process and execute. The "shatter attack", a term coined by Paget, works as follows: first, the malware copies the code-to-be-injected to the clipboard. Next, it sends `WM_PASTE` message to target process to paste the clipboard contents into a text field on the GUI of the target process. At this point, the malicious code has been moved onto the memory space of the target process. To execute this code, the malware process sends another window message, a carefully crafted `WM_TIMER` message, which causes a jump to the address of the malicious code. The main difference with our attacks is the presence of the malicious code during the injection, while with *Cut-and-Mouse* we use and control a privileged application as a "puppet" to perform various operations without injecting new code into the target process memory.

Comparison to Previous Attacks

Cut-and-Mouse and *Ghost Control* are two novel attacks on AVs, of which the main principle is to simulate user commands by programmatically synthesizing mouse and keyboard events. As we reviewed above, there are other techniques in the literature which shares similar behaviour. Table 5.2 illustrates the characteristics of these attacks and compares to that of *Cut-and-Mouse* and *Ghost Control*.

Table 5.2: Comparison of *Cut-and-Mouse* and *Ghost Control* to Relevant Attacks that Synthesize Window Messages.

Characteristics	Ghost Control	Cut-and-Mouse	Synthetic Clicks [104]	Shatter Attack [107]
Exploits a Bug in OS	No	No	Yes	No
Modifies Target Process	No	No	No	Yes
Utilizes Clipboard	No	Yes	No	Yes
Requires a Text Edit Field	No	Yes	No	Yes

First of all, *Ghost Control* attack does not require a bug to exist in the OS, instead, it targets the applications that do not use the privileges provided by the OS, similar to Shatter Attack. In contrast, Synthetic Clicks [104] depends on a bug in the OS. Second, differently from Shatter Attack, which performs arbitrary code execution, *Ghost Control* only uses the functions exposed within the GUI of the target application. Therefore, *Ghost Control* leaves no trace in memory, while Shatter Attack modifies the target process and leaves artifacts that can be detected in the memory dumps. In a sense, *Ghost Control* attack can be considered as puppeteering the target application while Shatter attack is more close to poisoning the target. That said, *Ghost Control* needs exact coordinates of the screen to successfully work, while Shatter Attack is independent of the target system's display.

Similar to Shatter Attack, *Cut-and-Mouse* utilizes system clipboard and needs the target application to have a text field. However, similar to *Ghost Control*, the impact of *Cut-and-Mouse* attack is also limited to the functionality of the target applications, i.e., `Notepad` and `Paint`. Consequently, *Cut-and-Mouse* technique is naturally suited to damage files and, therefore, can be exploited to perform ransomware, wipeware or similar destructive attacks.

Previous Research on Security of AVs

Traditionally, AVs have been in the target of security researchers due to their incomparable importance. Since AV vendors mostly utilize blacklisting as the main defense technique, many researchers investigated this area. For instance, [108] and [109] analyzed the feasibility of evade detection via obfuscation. Another significant research topic about AVs is the implementation related vulnerabilities. To name a few examples: [84, 110–113]. That said, the discoveries in this field mostly involve the bugs in the AV software, rather than a flaw in their design or threat model. Finally, in [81], Al-Saleh and Crandall developed a technique to determine if the target AV is up-to-date using side channel analysis, allowing the attacker to learn which signatures exists in virus database of the victim.

5.8 Discussion

Secure composability is a well known problem in security engineering. It challenges developers to ensure that security properties enjoyed by individual software components are preserved when the components are put together. It also challenges them to demonstrate that the components together give stronger security assurances than just the mere sum of their original properties. This rarely happens in practice, and the opposite is quite often true. Components that, when taken in isolation, offer a certain known attack surface do generate a wider surface when integrated into a system. Intuitively this seems obvious. Components interact one another and with other parts of the system create a dynamic with which an attacker can interact too and in ways that were not foreseen by the designer. An attacker can, for example, uses a component as an oracle or replay its output to impersonate it while interacting with another.

This is exactly what we have found happening to mechanisms like UIPI and AV software. They provide a robust defense when tested individually against a certain target, but the attacks that we demonstrate in this chapter show that their combination reveals new vulnerabilities. We draw two considerations from it.

First, in complex systems it is essential to control the message-flow between security critical components. This is actually enabled by Microsoft via UIPI. It allows messages flowing from sender applications to receiver applications only when the integrity level of the first is not less than the integrity level of the second. In principle, UIPI enables a good defence mechanism, but the problem is that integrity levels do not reflect trust: they merely indicate when an application runs with administrative right (high), in standard mode (medium), or in a sandbox (low). The authority who decides which level an application takes is generally the operating system, and sometimes the user, after a request from the application. It may be, like in the scenario that we illustrated in Section 5.4, that developers do not implement that request.

This is against what Microsoft Driver Security Guidance suggests [114]: "*It is important to understand that if lower privilege callers are allowed to access the kernel, code risk is increased. [..] Following the general least privilege security principle, configure only the minimum level of access that is required for your driver to function.*". We think that the process which controls the status of the anti-malware and AV's kernel module should be designed to require 'high' IL. Our findings show that several anti-malware companies either failed to follow this guidance or have misjudged the minimum level requested for their security, or did not diversify enough between kernel and non-kernel modules.

Secondly, and this is linked to our finding in Section 5.3, relying only on integrity levels is not sufficient to ensure system security. This does not surprise, since UIPI has been designed to protect processes, and in fact anti-malware applications top-up their defence strategy relying on whether an application is whitelisted, that is, *trusted*. Only trusted applications can e.g., access protected files. But, our findings have revealed a dissonance here: medium integrity level applications, like `Notepad`, are considered trusted and thus allowed to e.g., access protected files. But an application with medium integrity level, that is running with standard user rights, does not necessarily behave in a benign manner. As we showed for the case of `Notepad`, medium but untrusted applications, such as malware, can have their actions looking like be trusted by using the application as a puppet; in so doing, they can bypass the anti-malware guard.

We think that a better defence is to combine the integrity levels and the trust label used by anti-malware. We state it as the following principle:

Security Principle 1 *Messages between applications should be allowed only when the sender has at least the same integrity level as the receiver* and *and the sender is at least as trusted as the receiver.*

Principle 1 reminds the renowned Bell and La Padula Model on messages-flow between different security "clearence" levels [115] (see also [116]). But it is not exactly the same, since we cope with "security" instead of confidentiality. Attempting a formalization of Principle 1, components should be classified by "security levels", made of two elements: the UIPI "integrity levels", (I = [admin , user, or sandbox], ordered) and the anti-virus software's "trust levels" (T = [digitally signed / whitelisted, not digitally signed / not whitelisted], also ordered). Principle 1 suggests a policy saying that an application of security level (I, T) should not accept messages coming from applications of security level (I', T') when $(I' < I)$, or when $(I' \geq I)$ but $(T' < T')$.

In conclusion, we believe that applying Principle 1 would have prevented receiving `SendInput` from effecting whitelisted applications that has a potential to be exploited, e.g., `Notepad`. One should, however, evaluate whether this may also broke some of the existing automation software solutions. A conclusive statement about this would require to perform a wide spread test on automation applications. It also had fostered AV vendors take measures not only to protect the system, but also to protect their AV programs against other supposedly trusted applications, in addition to conventional malware attacks against AV products. A practical fix is to configure AV kernel module to require admin rights to be accessed. In this regard, it might be helpful to monitor `SendInput` API and block all simulated keyboard and mouse events dispatched to AV program although the problem of understanding whether a low-level event, such as an interrupt, has been generated by a human or not might be difficult to solve in general.

Sandbox Evasion | 6

More than ever there is a need of automated systems to perform malware analysis as new malware samples are created at such a pace that security analysts are unable to manually analyze them. For instance, in the first 9 months of 2020, more than 1 billion samples were found on-the-wild, with an average of 350,000 new malicious program released each day in the latest months of the year [117].

Analysing new samples of malware is a complex task. In fact, on one hand, security researchers are aimed at developing malware analysis systems, such as ”sandboxes”, i.e. confined and protected systems (typically based on virtualisation) which are used to execute suspected untrusted programs to analyse their behaviours. On the other hand, cyber-criminals creating malware and associated infrastructure want a return on their investments in their effort, and therefore are becoming more adept at developing threats that can evade increasingly sophisticated sandboxing environments [118]. In particular, in recent years, several ransomware families have started incorporating advanced evasion techniques [119]. For example, the first version of the `WannaCry` ransomware included a basic anti-sandboxing mechanism, based on DNS query, to bypass analysis [120]. Evasive techniques employed by malware are based upon the fact that the execution environment of a process may change slightly depending on whether it is run inside a native or a virtualised/sandboxed environment, e.g. due to the presence of memory and execution artifacts introduced by the sandboxing environment (environment-based evasion) or due to the different execution time of the process inside a sandbox (behavioural-based evasion) [121]. Malware may also specifically look for signs of emulation or virtualisation, e.g. specific drivers created inside a virtual machine, or the presence of specific applications, such as standard add-ons used by sandboxing solutions, or the lack of standard application and files (e.g., an empty virtual machine) to detect sandboxes. Finally, certain malware will check for user input before performing any action, and the lack of user interaction will sometime cause the malware to infer it is being run in a sandboxed environment. In all these cases, the goal of any evasive malware is to detect whether it is being run in a sandbox and, in such a case, stop performing any malicious action to avoid being analyzed further by sandboxing systems – therefore making it hard for analysts to extract the features and describe the behaviour of malware for future detection.

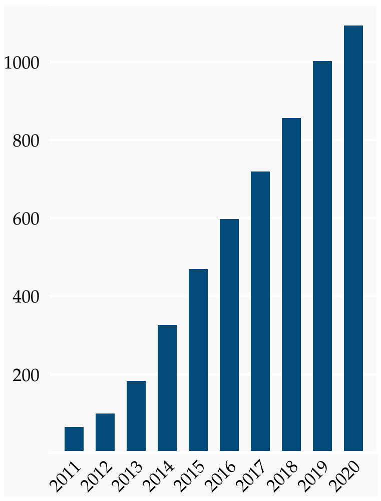


Figure 6.1: Total number of known malware samples by years. As of September 5, more than 1 billion samples registered in 2020 [117].

Other works have addressed the problem of the detection of evasive malware. For instance, [122] proposes a reliable and efficient approach to detect malware with split personality – behaving differently according to the running environment, in an attempt to evade analysis. Similarly, [123] describes BareCloud, an automated evasive malware detection system which is based on bare-metal dynamic malware analysis to compare the traces of analysis on different environments and compare them with the bare-metal one – a discrepancy meaning the malware is employing evasive techniques. Finally, [124] presents MalGene, an automated technique for extracting analysis evasion signature by leverages bioinformatics algorithms to locate evasive behavior in system call sequences.

In this chapter, we describe a technique to bypass sandboxes that we have found being used by an active ransomware that we inspected. We also describe the process that has allowed us to observe the malicious activity in this specific sample. To the best of our knowledge, this malicious technique has never been described before: it is stateless, i.e., the attack comprises multiple phases but the malware does not store any data on the target machine, and does not depend on any logical condition on the victim system to be triggered. The technique enables the ransomware to employ a stealthy attack strategy which would evade detection by sandboxes. After presenting our findings, we discuss the feasibility and impact of cyber attacks that use this technique, and propose possible mitigation strategies.

6.1 Methodology

The detection methodology we follow in this chapter is composed of the following steps: firstly, we run a malware sample multiple times in a sandbox and collect the traces. Afterwards, we check if the sample has performed no suspicious activity in the first run, but has acted maliciously in the subsequent runs. If this is the case, it means the malware is empowered with some evasive functionalities. In the following, we explain the details of (i) the steps we have followed when building the test environment, (ii) how we have gathered the data set and (iii) how we have conducted the experiments.

The analyses are performed using Cuckoo Sandbox, an open source automated malware analysis system [125]. Our test environment consists of 20 VMs running atop Kernel-based Virtual Machine (KVM)[1], each has 2 CPU cores clocked at 2.60 GHz and 2GB RAM. On each VM, we performed a clean installation of Windows 7 32-bit OS with SP1. No guest agent is installed on VMs. Moreover, we performed additional steps to ensure the virtualization artifacts are removed to make the detection of the sandbox harder [126]. For instance, default BIOS[2] of VMs is replaced with a customized version which contains a real-world vendor name. Likewise, the hypervisor flag of the virtual CPUs is disabled to prevent guest OS from being aware of the virtualization. Next, we created user profiles on VMs and installed popular applications such as web browser (along with top-rated extensions), multimedia player, archive utility and office software. Furthermore, we artificially populated usage history in these profiles to reflect an authentic user. Finally, we took the snapshot of VMs and configured the Cuckoo accordingly.

1: Kernel-based Virtual Machine, https://www.linux-kvm.org.

2: SeaBIOS, https://seabios.org/SeaBIOS.

Ransomware samples are obtained from the malware corpus provided by VirusTotal [127]. To filter cryptographic ransomware, i.e., the programs that encrypts victim's data, we performed a query using ransomware-related keywords in the anti-virus scan results, such as *ransom*, *crypt*, and *lock*. After the search is complete, we had a set of 112 potential ransomware samples.

Once the data set is ready, we submitted the ransomware samples to Cuckoo Sandbox to study their behaviours. The label of VM used for each analysis task is noted to use in the second run. After all samples are analyzed, the samples that did not show any malicious activity are re-submitted selecting the same VM. Finally, we compared the reports generated by Cuckoo Sandbox to determine the samples that did not perform encryption in the first execution, but encrypted the victim's files in the second run.

6.2 Results

Out of 112 malicious samples, one ransomware sample showed no malicious activity in the first execution, but encrypted user's files in the second run. SHA256 digest of the sample is `bcbc1aee86f5e1fdc2ba6fcb2e2993 3933b132a4c3d0f2eb0f73061702041243`. As of August 2019, 58 out of 66 anti-virus engines at VirusTotal identified this sample as a malicious program. Malware labeling tool AVClass [128] identified the sample as a `TeslaCrypt` variant.

Behavioral Analysis Reports

Now, we compare the behavioral analysis reports of two executions of the ransomware sample, generated by Cuckoo Sandbox.

First Execution

In the first execution of the sample, we observed no write operation on user files. Although no persistent change were made to the files, the sample

- called `GetComputerName` API to retrieve the NetBIOS name of host;
- called `GetVolumeInformation` API to collect various information about the virtual volumes and the hard disk;
- read the `InstallDate` key[3] in the registry which stores the installation date of the OS; and
- read `MachineGuid` key[4] from the registry, which is created during the installation of Windows OS.

3: `HKML\SOFTWARE\Microsoft\Windows NT\CurrentVersion\InstallDate`

4: `HKML\SOFTWARE\Microsoft\Cryptography\MachineGuid`

Up to this point, the collected pieces of information would give the attacker the ability to identify the victim's computer with a high accuracy, i.e., generate the fingerprint of the machine.

Immediately after taking the fingerprint of the environment, the ransomware sample established several network connections. Table 6.1 represents the IP addresses and the domain names that the sample connected using the HTTP protocol only. Using the threat intelligence services, we were able to confirm that the URL of each connection is related to a well-known, malicious activity pattern previously reported by anti-malware community.

Despite the suspicious activity summarized above, the sample did not perform any encryption, and shortly after the execution, it terminated

Table 6.1: HTTP connections of the analyzed sample.

IP Address	Domain Name
`204.11.56.48`	`imagescroll.com`
`85.128.188.138`	`stacon.eu`
`109.73.238.245`	`surrogacyandadoption.com`
`69.89.31.77`	`biocarbon.com.ec`

itself, leaving no artifacts on the analysis environment. As a result, the malice score assigned to this sample by Cuckoo Sandbox is 2.4.

Second Execution

In the second run, the sample conducted the same reconnaissance steps taken in the first run. However, this time, after fingerprinting and connecting to remote addresses, and instead of ceasing, the sample

- silently deleted VSS copies (backup copies or snapshots of files or volumes, created by Windows OS) – this operation is typically encountered in ransomware attacks to prevent recovery of files;
- dropped an executable file – this is another typical malware action to bypass signature based detection;
- configured registry in order to automatically run itself at each login; and
- created encrypted files, rename them by appending the extension `.mp3`, and deleted the original files.

Cuckoo developers state that there is no upper limit for malice score, but currently the security threshold is set to 10. After the second run, the analyzed sample scored 25, and hence labeled by Cuckoo as "very suspicious".

Reconstructing the Attack Scheme

The second behavioral analysis report displays a detailed picture of the attack. However, one piece of the puzzle is still missing: how does the ransomware sample decides to commence the attack?

Since the analyses are performed on freshly restored snapshot of VMs, the sample cannot store any state on the VM. Namely, all data generated during an analysis will be lost once the analysis ends. Furthermore, the ransomware is analysis-aware and collects various information that can be used to fingerprint the environment. Moreover, the sample connects to several remote machines before starting the attack, if it attacks.

Algorithm 8 Server-Cooperated Attack Strategy.

```
1: function ATTACK
2:     machineID ← GenerateFingerprint()
3:     SendToServer(machineID)
4:     response ← GetResponseFromCC()
5:     if response == CEASE then ExitProcess()
6:     else EncryptFiles()                    ▷ response is ATTACK.
7:     return Success
```

A full disclosure of the sample's logic is possible only if we reverse-engineered the sample's executable. We did not, but we can formulate an hypothesis on the sample's possible functionality from its behavioral analysis. For the sake of the goal of our discussion, this suffices. Surely, once infected the victim machine, the sample collects information likely to identify the running environment and generate a fingerprint of the machine. And it is almost certain that the information is sent to a C&C server. Here, we speculate. The C&C may use the fingerprint to identify the victim machine and to decide whether the sample is executed for the first time on it. In this case, we keep on speculating, the C&C server returns a `CEASE` message to tell the ransomware to stop all operations and remain silent. On the contrary, when the victim machine is known, the C&C server sends an `ATTACK` message to trigger the ransomware. If we are right, the attack occurs like as in Algorithm 8 and its information flow as in Figure 6.2.

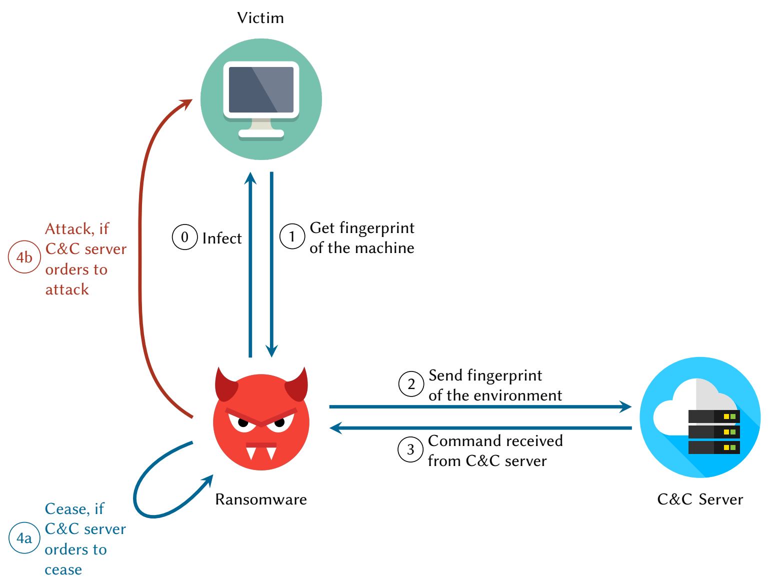

Victim
4b Attack, if C&C server orders to attack
0 Infect
1 Get fingerprint of the machine
2 Send fingerprint of the environment
3 Command received from C&C server
Ransomware
C&C Server
4a Cease, if C&C server orders to cease

Figure 6.2: Reconstructed attack diagram.

6.3 Discussion

In this section, we discuss the security impact of the described evasion technique and propose potential mitigation strategies.

Evading Sandbox Detection. The evasion technique analyzed in this chapter allows the malware to stay under the radar, at least in the first run. This might allow the malware to bypass a defense system, for example, a system that whitelists applications that do not show malicious activities when run in sandbox environment. It should be noted that, though, the ability to evade sandbox detection comes with a price: malicious functionality of the malware gets activated on if the user runs the malware executable as many times as configured by the malware authors, i.e., typically more than once, which is not guaranteed to happen.

Bypassing Malware Appliances. In [129], authors report that three popular malware appliances from well-known vendors analyze the binaries in an isolated environment, i.e., run the executables in a machine disconnected from the network. In this setting, the malware would not be able to connect the C&C server and therefore would not receive an ATTACK command. Consequently, the binary would not exhibit any malicious behaviour. In this case, it is likely that the appliance is left with only signature based detection and the static analysis options which both can be evaded via obfuscation. In the end, the malicious binary would be not detected by these malware appliances. We note that, this result is not exclusive to the attack technique described here, rather, it is the limitation of the isolated analysis strategy followed by these appliances.

Mitigation Strategy. As a countermeasure to this attack, we propose the following strategy: to reveal the malicious behaviours, the analyzed samples should be executed multiple times in each analysis session. This way, the malware will run at the same environment for more than once, which will increase the chance of showing its real behaviour. Of course, the malware authors may set a higher threshold to prevent detection, but this would also delay the attack which is against the goals of cybercriminals. Still, a high threshold would prevent detection by the sandbox. However, even if the detection fails after N execution, i.e., the malware do not show a malicious activity, and the malware passes sandbox and reaches the actual user environment, the user –at least– would be secure for the first N execution. That said, our analysis assumes that the C&C uses a basic counter and a threshold to decide attack, but it may also utilize a smarter decision algorithm. For example, C&C might ignore subsequent fingerprint messages if the time frame between two executions are too narrow. Finding the best move of C&C in this game is therefore an open problem.

Defense in Depth. Alternatively, we propose a defense-in-depth strategy to be used in user environment as an auxiliary mitigation for this attack. To recall, the analyzed technique aims to bypass sandbox detection by acting benign in the first run. That is, if the each run of the malware executable occurs on an environment different from previous ones, the executable would not cause any damage. Using this idea, one can make the actual user environment look unique to unknown/untrusted executables for each run. This way, the executable –assuming that it is a malware– would report to its C&C server a unique fingerprint at each execution. Once received a previously unseen fingerprint, C&C server would not find any match in its database, and send a CEASE message to the malware. It should be noted that realization of this defense method is feasible. For instance, it can be achieved by hooking certain API that can potentially be used to fingerprint the environment and randomize their return values. However, fingerprinting is a practice also used by software companies, e.g., for activating proprietary programs. Therefore, benign applications should be whitelisted when applying this defense method to prevent undesired interference.

Network Dependency. The investigated technique involves communication between a C&C server, to receive ultimate decision to attack. This obviously requires the remote server to be available when the malware program is executed each time. If the C&C server becomes unreachable, there would be no location to keep the state of the attack, and the strategy cannot work: the malware would not perform its nefarious actions. At first, this might look like a shortcoming of the technique, as blocking malicious IP addresses is an efficient and effective practice to distort malware communications. However, malware authors has been encountering this limitation for a long time, and they already employ workarounds. For example, as analyzed in [61], authors of Cerber ransomware uses messages encoded in Bitcoin transactions to coordinate the C&C servers. Being decentralized, impeding communication with blockchain network is considered difficult. Malware authors can enhance the examined evasion technique by integrating the blockchain-based coordination of C&C servers. We therefore argue that the said limitation do not decrease the significance of the threat.

Prevention of Cryptographic Ransomware

Stopping Ransomware by Controlling CSPRNGs 7

In this chapter, we present our anti-ransomware system UShallNotPass. Our solution has an *ex ante* nature, that is, it attempts to prevent a ransomware from encrypting files in the first place. UShallNotPass is the first *cryptographic ex-ante* defense against ransomware to the best of our knowledge.

To achieve our research goal, we approach the problem from a cryptographic perspective and propose a strategy which relies on two fundamental observations. First, the keys-for-money exchange on which ransomware based the success of their business works only if the victim has no other ways to recover the files but paying for the release of the key. To achieve this goal, a ransomware must properly implement *strong cryptography*, which means:

- *robust encryption*, that is, well-established encryption algorithms;
- *strong encryption keys*, that is, long and randomly generated strings.

Second, if these are the tools that ransomware necessarily need, one can try to prevent any unauthorized use of them. Nothing can be done to prevent ransomware from using robust encryption: these algorithms are public and ransomware can implement them directly. Thus, we have to focus on denying access to a source of strong encryption keys.

7.1 Requirements

The requirements that inspired and, a posteriori, characterize the security quality UShallNotPass (we use here the terminology as suggested by the RFC 2119 [130]):

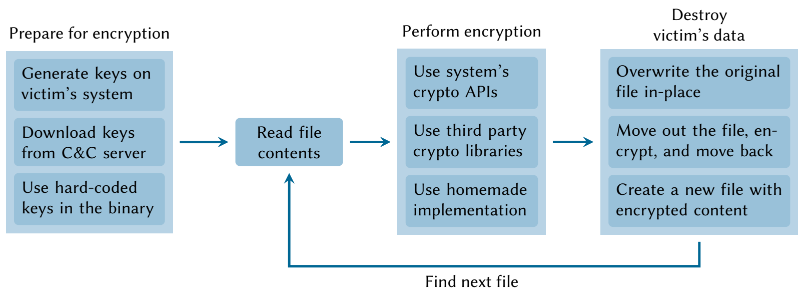
(R1) it MUST stop all currently known ransomware;
(R2) it SHOULD be able stop zero-day ransomware;
(R3) it MUST NOT log cryptographic keys and thus:
  - it should not introduce the risk of single point of failure that smarter ransomware can try to break;
  - it should not endanger the level security of benign applications (e.g., TLS session keys);

(R4) it SHOULD be easily integrated in existing anti-virus software, OSs, and access control tools;
(R5) it MAY be implemented directly in an OS's kernel or in hardware.

7.2 On Ransomware and Randomness

We answer a fundamental question: why does ransomware need random numbers and from which sources it must necessarily obtain them? The answer will help to understand the rationale of UShallNotPass's *modus operandi*.

As any other software virus, a *ransomware*, say *R*, runs in the victim's computer. On that machine, *R* finds the files, *F*, that it will attempt to encrypt. To work properly *R* needs two tools: a robust encryption algorithm and a means to create strong key, *k*. With those tools, *R* has all it needs to encrypt *F*. The encrypted files will replace *F* irreversibly and irremediably until the ransom is paid, triggering the release of the decryption key. The diagram in Figure 7.1 shows this simple work-flow in picture with some detail that we are going to discuss.



Prepare for encryption
Generate keys on victim's system
Download keys from C&C server
Use hard-coded keys in the binary
Read file contents
Perform encryption
Use system's crypto APIs
Use third party crypto libraries
Use homemade implementation
Destroy victim's data
Overwrite the original file in-place
Move out the file, encrypt, and move back
Create a new file with encrypted content
Find next file

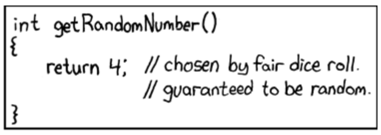
Figure 7.1: Generic Ransomware Functionality.

First let us see how *R* typically acquires the tools it needs. Strong encryption algorithms are publicly available. Current ransomware just makes use of those robust encryption algorithms, either by statically linking third party networking and cryptographic libraries, for example the NaCl or the OpenSSL or by accessing them via the host platform's native APIs.

However, to obtain strong keys, *R* has to access *secure* randomness sources. *R* has a few alternatives for doing so, but only one is secure. In fact:

(1) *R* can have a strong *k* hard-coded, precisely in a section of its binary code, but this solution leaves *k* exposed. *R* can be probed and have *k* can be extracted from it e.g., by Binary Analysis.

(2) *R* can download a strong *k* from the Internet. Occasionally ransomware samples employ this technique and download encryption keys from their C&C servers, but also this option exposes the key. It can be eavesdropped e.g., by an Intrusion Prevention System (IPS). Note that although ransomware will likely use secure communication (i.e., an encrypted channel), the problem of establishing the session key remains, looping the argument (e.g., if the key is hard-coded in *R*, there is a way to reverse engineer it, *etc*).



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

Figure 7.2: Incidents related to random numbers are famous in information security chambers. *Courtesy of* `xkcd.com`.

The remaining alternative is to let *R* generate its own *k*. But for *k* to be strong, *k* must be randomly chosen from a data set with sufficient entropy to make brute-force attacks infeasible and be kept safe. Where, in a computer, *R* can find that randomness it requires to build strong keys? True randomness is generally unavailable and thus ransomware must resort to those few deterministic processes that return numbers which exhibit *statistical* randomness. These processes are known as *Random Number Generator (RNG)* functions. *R* can implement them. But, being deterministic algorithms, RNGs are always at risk to be error-prone. If they produce predictable outputs the cryptographic operations build on them cannot be considered secure [131], because with a predictable "randomness" all hybrid encryption schemes would be vulnerable to plain-text recovery [132]. History proves that this concern is legitimate. To give a few examples, in the Debian–OpenSSL incident, RNG was seeded with an insufficient entropy which resulted generation of easily guessable keys for SSH and TLS protocols [133]. Moreover, Dual Elliptic Curve Deterministic Random Bit Generator (Dual EC DRBG) of Juniper Networks found to be vulnerable, allowing an adversary to decrypt the Virtual Private Network (VPN) traffic [134]. These incidents shows that extreme care should be taken when dealing with randomness. *Non-cryptographic* random number sources have weaknesses [135], and they should be avoided in cryptography.

The safest way (i.e., the way to avoid that risk of being error-prone in generating pseudo random numbers) is to use well tested and robust functions called *Cryptographically Secure Pseudo-Random Number Generator (CSPRNG)*.

In the OSs of the Microsoft (MS) Windows family, CSPRNG functions are available through dedicated APIs (analogous solutions do exist in other OS families, although the name of the functions will change). User mode applications call cryptographic APIs to get secure random values of desired length. Historically, Windows platform has provided the following APIs:

- `CryptGenRandom`: Appeared first in Windows 95 via Microsoft CryptoAPI (MS CAPI), now deprecated.
- `RtlGenRandom`: Beginning with Windows XP, available under the name `SystemFunction036`.
- `BCryptGenRandom`: Starting with Windows Vista, provided by Cryptography API: Next Generation (CNG).

Legacy applications call the function `CryptGenRandom` to obtain a random value or, as modern applications do, call `BCryptGenRandom`. When developers do not need a context, they can also directly call `RtlGenRandom` to generate pseudo-random numbers. Moreover, `CryptGenRandom` internally calls into `RtlGenRandom`. While the implementation of `RtlGenRandom` is not open-sourced, a relevant documentation [136] states that various entropy sources are mixed, including: (i) the current process ID; (ii) the current thread ID; (iii) the ticks since boot; (iv) the current time; (v) various high-precision performance counters; (vi) an MD4 hash of the user's environment block, which includes username, computer name, and search path; (vii) high-precision internal CPU counters, such as RDTSC, RDMSR, RDPMC; and (viii) other low-level system information[1].

1: For the complete list of entropy sources, please refer to [136, p. 262].

To ensure the strength of the encryption key, modern ransomware takes advantage of the CSPRNG functions that the host OS provides.

Note that, for the same reason, those functions are also used in: (i) Initialization Vectors (IVs): used by both stream and block ciphers; (ii) salts: used in Key Derivation Functions (KDFs); and (iii) paddings: block ciphers (in Electronic Codebook (ECB) or Cipher Block Chaining (CBC) modes of operation) and public key encryption algorithms.

7.3 USʜᴀʟʟNᴏᴛPᴀss' Rationale

From these considerations it should be clear why the central idea of this research is guarding access (i.e., of intercepting incoming calls) to (the

APIs of) CSPRNG functions and why it works: any strong ransomware must employ a CSPRNG to create secure keys. Keys generated by alternative methods may not be so strong and files encrypted with them could be decrypted, e.g., by using a decryptor from No More Ransom project (see Chapter 1).

Thus it should be clear that CSPRNG functions are *security-critical resources*, and hence only authorized processes should have access to them. This means that deciding which processes should be authorized is critical, but is not within the scope of this thesis and it will be addressed in future work. Generically speaking, we suggest that authorized applications are those which have been *whitelisted* or *certified*. The process of authorizing an application can be as simple as let the user (or the system administrator) decide about whether s/he trusts the application (e.g., as done by [23]), or it can result from an agreement protocol between the OS vendors (e.g., Microsoft, Apple) and the developers of cryptographic applications. Whatever the strategy, similarly to what happens in Europe about applications that process personal data, application developers have to gain their authorization/certification. Ransomware, developed for the illegal software market, should therefore be excluded. The third and last consideration is that we suggest that unauthorized requesters of *CSPRNG functions* are terminated.

Thus UShallNotPass prevents ransomware damaging files in the system and no recovery is necessary. In Section 7.6 we will see how this strategy is essential for stopping `NotPetya`.

Listing 7.1: Outputs of non-cryptographic PRNGs can be reproduced, therefore the keys derived using them be obtained later. The following example illustrates usage of two `Random` objects instantiated with the same seed value in C#. Both objects generate integers between 0 and 100. N.B., they produce the exact same outputs.

```
int seed = 352;
var rnd1 = new Random(seed);
var rnd2 = new Random(seed);

Console.Write("First: ");
for(int i = 0; i < 5; i++){
   int num = rnd1.Next(100);
   Console.Write("{0} ", num);
}

Console.WriteLine();

Console.Write("Second: ");
for(int i = 0; i < 5; i++){
   int num = rnd2.Next(100);
   Console.Write("{0} ", num);
}

// [On Console]
// First: 61 10 81 69 12
// Second: 61 10 81 69 12
```

Assumptions UShallNotPass targets ransomware families that follow secure development strategies and utilize strong cryptography. We will deal only with the strongest amongst current ransomware, that is, we ignore insecurely designed and badly implemented ransomware families, for instance those which call non-cryptographic PRNGs to generate keys (see Listing 7.1) or those which encrypt files with home-brew algorithms. For these ransomware we already have solutions able to mitigate their effects.

Currently, UShallNotPass runs as a software component of the host and relies on the security guarantees of the host OS. Therefore we assume that the OS on which our system runs is up-to-date. In particular, we require that ransomware does not exploit any zero-day vulnerabilities to escalate privilege. It should be noted that this requirement is inherent to every protection software running on any OS. Furthermore, an outstanding feature of our strategy is its being obfuscation agnostic, i.e., UShallNotPass targets all ransomware samples from non-obfuscated to highly-obfuscated ones.

7.4 UShallNotPass' Design

We now describe the inner mechanism of our system in more detail. The architecture of UShallNotPass and the workflow is depicted in Figure 7.3. It has two separate components: Interceptor, and Controller. Interceptor captures the calls made to `CryptGenRandom` API (a CSPRNG offered by

Windows OS) and dispatches the process ID to the Controller, which searches the Whitelist DB to decide whether to allow or deny access. No parameters or outputs are logged.

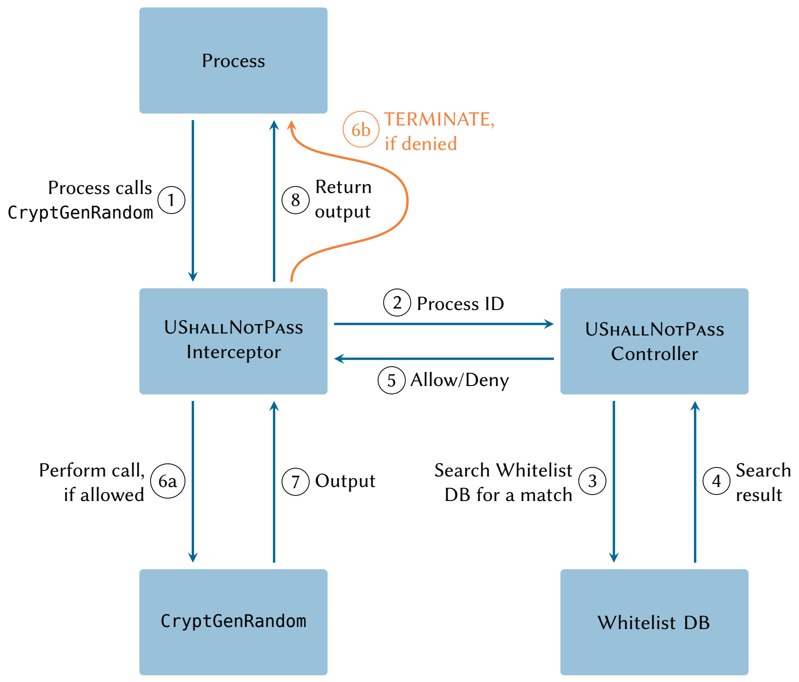


Figure 7.3: Architectural view of UShallNotPass [44]. When `CryptGenRandom` API is called, Interceptor identifies the caller and dispatches the process ID to Controller. If the application is authorized, the call is executed and the result is returned to the caller. Otherwise, the call is blocked and the caller process is terminated.

High Level Description

Essentially, UShallNotPass is an access control mechanism over the CSPRNGs of the host system: it intercepts calls to CSPRNG functions and queries the process ID of the caller. Once the caller process is determined, UShallNotPass reaches a decision according to a *system policy*. For example, the system policy might enforce the following configuration. If the caller process is authorized, that access is granted and it obtains the secure pseudo-random number. Otherwise, UShallNotPass blocks the call and terminates the caller process. In our prototype implementation, we used this system policy.

Intercepting Requests to CSPRNG

As we argued in Section 7.2, ransomware requires to use CSPRNG of the host system. In the current architecture of modern OSs, there are limited number of resources which provide cryptographically secure pseudo-random numbers. It is feasible to intercept the calls made to CSPRNG functions of the host system and redirect the control to the decision making component of UShallNotPass.

System Policy & Managing Access Control

When a request is made to access the CSPRNG of the system, to reach a decision to grant or deny access UShallNotPass follows a *system policy*, a set of rules, for instance, determined by the system administrator. The system policy can be specified in various ways, depending on the needs and the nature of the host system. Our current design implements it as a *whitelist*, i.e., list of applications allowed to access CSPRNG which a system administrator determines immediately after UShallNotPass is installed. It can be more complex thought, such as determined by the OS vendors in agreement with developers of cryptographic applications and based on accreditation, granted after established security checks.

Further security measures can be necessary. Here we mention two in particular:

- *Digital signatures*: Code signing is a technique to verify the integrity of the executable and the origin of the source. Digitally signed software has therefore higher trust score when evaluated by anti-malware products and OSs. For example, Microsoft uses Authenticode [137] to verify the signature of the executables and kernel drivers. Following the same approach, we design UShallNotPass so that it can be configured to allow applications with digital signatures to access to CSPRNG functions of the host system.
- *Dynamic Decision*: It may be desired to have a minimal whitelist, and extend it when necessary. So that when an application requests a cryptographically secure pseudo-random number for the first time, it is put on hold and the decision will be made on that time. A similar measure has been described in [23, 26], but it involves the user. Considering this choice unsafe, UShallNotPass instead interacts exclusively with the administrator. UShallNotPass's system policy can be set to force to ask the exclusive permission of the system administrator when an application calls a CSPRNG function for the first time.

Once the whitelist is created, UShallNotPass will start intercepting the access requests to CSPRNG of the host system. For each request, identity of the caller will be determined and UShallNotPass will decide whether to grant access. If the result is positive, the process is allowed to obtain the pseudo-random number. Otherwise, the request is blocked and the process is terminated.

Needless to say, it is therefore of uttermost importance to secure the system policy itself from unauthorized modifications (e.g., store it in the form of a file under a directory accessible only with administrator privileges).

7.5 Implementation

We implemented a prototype of UShallNotPass which targets Windows 7 OS. On Windows 7, user-mode processes mostly invoke `CryptGenRandom` API to get cryptographically secure pseudo-random numbers. Therefore, our implementation intercepts each invocation of `CryptGenRandom` API and determines the identity of caller process. To this end, UShallNotPass consists of two components:

- *Interceptor* which intercepts calls made to `CryptGenRandom` API, collects and transmits the identity of the caller process to Controller, and takes the appropriate action that Controller commands.

- *Controller* which gets information from the Interceptor and returns grant/deny commands according to the system policy.

Intercepting Calls to CSPRNGs

There are various ways of intercepting calls on Windows platform, including patching System Service Dispatch Table (SSDT), modifying Import Address Table (IAT) and injecting a DLL to target process. We followed the DLL injection technique and used Detours library of Microsoft Research for this purpose. The Interceptor of UShallNotPass is hence a DLL module which is loaded into target process on the system. For ease of prototyping, we load the Interceptor into processes using *AppInit DLLs* technique [138]. Once loaded, it hooks `CryptGenRandom` function, that is, whenever `CryptGenRandom` is called by a process, program flow is routed to the Interceptor.

Decision of Authorization

The Interceptor calls `GetModuleFileName` to obtain the full path to the module of the caller process, which can point to a DLL or an executable. The file path information is passed to the Controller, whose response is forwarded to the Interceptor. Controller computes the SHA256 digest of the binary file of the module and checks whether it is in the whitelist.

If the result is positive, a GRANT command is returned to the Interceptor, or a DENY command otherwise. Once the decision is received from Controller, Interceptor executes it. If the decision was to grant access to secure random API, Interceptor calls `CryptGenRandom` with the intercepted parameters and returns the result and control to the caller process. If the decision of Controller was to deny the request, then Interceptor calls `ExitProcess` function, which causes the caller process to end[2].

2: Calling `ExitProcess` can as well cause process to crash, which, eventually ends it.

Maintaining the Whitelist

Whitelist is implemented as a file which contains the list of SHA256 digests of the binary executables. The integrity of the whitelist is protected by a keyed-hash value, appended to the end of the list. As another security precaution, the whitelist is located in a directory which only administrators has write permission.

Controller component of UShallNotPass has a GUI which provides the basic functionality to the user, such as adding an entry to the whitelist or removing one from it. Controller also logs relevant information about the call events to `CryptGenRandom` API, including time, SHA256 digest of the caller and the action taken.

7.6 Experimental Results

We tested our UShallNotPass with the aim to verify whether it complied with the requirements we stated in Section 7.1. Compliance with R3 does not need to be tested. It follows from the design: UShallNotPass does not store cryptographic keys (R3). Instead we test compliance with R1 and R2 indirectly by answering the following questions about UShallNotPass:

- **Q1** Does it stop ransomware before they encrypt any files?
- **Q2** Can it protect against zero-day ransomware?

Furthermore we are interested in knowing what is UShallNotPass's performance in time and space resources. A defense system that is not practical to deploy is considered useless.

- **Q3** What is the overhead of UShallNotPass in terms of resources?

The answer this third question gives evidence for compliance to R4 and R5: if UShallNotPass proves be efficient, it can be easily integrated with existing anti-virus software as an additional run-time control (R4). Its simplicity also suggests that controlling the access to critical functions can be implemented at least at level of OS kernel (R5).

We have not yet thought about the possibility to implement this mechanism at hardware level.

Experimental Setup

We conducted a series of experiments to test the robustness of UShallNotPass against cryptographic ransomware. We obtained real world cryptographic ransomware samples from well known sources including VirusTotal[3] and ViruSign[4]. In order to collect executables, we performed a search on these sources with the keywords *ransom, crypt* and *lock* which generally appear in the tags determined by submitters and antivirus vendors. Furthermore, we populated our collection by downloading samples from the links provided by Malc0de[5].

3: VirusTotal, https://www.virustotal.com

4: ViruSign, https://www.virusign.com

5: Malc0de, http://malc0de.com

Our initial test set had 2263 malware samples, each labeled as ransomware by anti-virus engines.

Collecting a malware sample is one thing, determining its type is another. A malware sample tagged "ransomware" may not necessarily be an active cryptographic ransomware. Therefore, we needed to check the obtained malware samples one by one and select the active cryptographic ransomware in order to build a valid sample set. For this aim, we utilized Cuckoo Sandbox,[6] open source automated malware analysis system. We created a VM atop KVM[7] and performed a clean install of Windows 7 OS. Next, we created a user environment on the VM and performed actions which reflects the existence of a real user, e.g., we installed various popular applications such as third party web browsers (and select plug-ins), office and document software, utilities etc. Moreover, we placed a number of files on the VM that typical ransomware families targets, such as office documents, images and source codes. When possible, we also removed traces of the virtualization, e.g., changed default device names of VM,

6: Cuckoo Sandbox, https://cuckoosandbox.org

7: Kernel-based Virtual Machine, https://www.linux-kvm.org/page/Main_Page tuning RDTSC, etc. Finally, we took the snapshot of the VM and finalized the configuration of Cuckoo for managing the VM.

After the test environment was set, we submitted the malware samples to Cuckoo which executed them one-by-one, on the clean snapshot of the VM. Although majority of ransomware samples attack the system immediately after infection, i.e., encrypts the victim's files, we allowed them to run 20 minutes unless the execution ends earlier. After each analysis, we inspected if any alteration/deletion of the decoy files observed on the test machine. We call a malware sample as an active ransomware if any of the decoy files is moved/renamed or has a new SHA256 hash after the analysis is completed. If Cuckoo does not detect any activity or hashes of decoy files are same until the timeout happens, we exclude the sample from our list of active ransomware.

To compare our results to the previous research, and to reason on the techniques used by malware authors, we identified the family of each ransomware sample. For this purpose, we employed AVclass [128], an automatic malware labeling tool which performs plurality vote on the labels assigned by AV engines.

We excluded the vast majority of the samples from our test set as they did not show up any malicious activity during the analysis. There might be several reasons behind this outcome. Firstly, it is a well known fact that malware authors try to avoid being analyzed and thus malware samples behave benign if they detect that they are run in a virtual environment. Ransomware authors also follow this strategy. Another reason of inactivity might be that the malware design may involve a C&C server which may be down for some reason. Finally, ransomware may require certain conditions met before start attacking, e.g., regional settings, wait for a specific date.

To sum up, we built a test set which contains 524 active samples from 31 cryptographic ransomware families to test against UShallNotPass.

Robustness

In this section, we will analyze the outcome of the experiments to find the answer of **Q1** and **Q2**.

To begin with, UShallNotPass stopped ransomware samples from *all families* in our data set, which includes famous and powerful ransomware families The details are reported in Table 7.1, where we also report for each family the average number of bytes per calls and the numbers of call, figures that support our argument that employing cryptographically secure pseudo-random numbers is a common property of all the ransomware.

Table 7.1 shows that UShallNotPass successfully stopped 94% of cryptographic ransomware in our test set, including WannaCry, Locky and TeslaCrypt and remarkably the unmitigated *NotPetya*. The remaining 6% of missed elements are false negative which may be of a few reasons. Firstly, our implementation of the Interceptor is not perfect. Therefore, *our* implementation might have missed to intercept those calls for some not obviously apparent technical reason. In fact, a dynamic analysis we performed on each representative for all the missed family (i.e., Cryptolocker, Filecryptor, SageCrypt and Yakes) has revealed that the ransomware actually invokes `CryptGenRandom`. Thus, in principle, they should have been stopped. Secondly, the samples might not be using CSPRNG and instead rely on the non-cryptographic PRNG functions that are not secure, and therefore not monitored by UShallNotPass.

Table 7.1: Measurements of CSPRNG usage. Next to *Family*, recalling the ransomware's family name, column *Sample* reports the number of elements in the family and the number of samples that UShallNotPass stopped. *CSPRNG Usage* column shows the need of using CSPRNG among ransomware and contains two subcolumns: *Bytes*, the average number of bytes that a sample of ransomware obtains from calling `CryptGenRandom`, and *#Calls*, the number of calls to the function.

Family	Samples (%)	CSPRNG Usage	
		Bytes	# Calls
Androm	7/7 (100%)	4125257	178
Bad Rabbit	1/1 (100%)	52	2
Cayu	1/1 (100%)	4216212	20261
Cerber	149/149 (100%)	22393	2786
Crilock	1/1 (100%)	3456637	15
Critroni	1/1 (100%)	4755304	392
Crowti	3/3 (100%)	5231466	14
Crypmod	1/1 (100%)	2167813	20118
Crypshed	1/1 (100%)	5137296	13
Cryptesla	8/8 (100%)	5125627	14
Cryptolocker	8/17 (47%)	2805603	10
Cryptowall	1/1 (100%)	2242370	10
Dynamer	2/2 (100%)	3954293	20118
Enestaller	3/3 (100%)	2127036	82
Enestedel	5/5 (100%)	3871449	61
Filecryptor	3/4 (75%)	64	1
Genkryptik	3/3 (100%)	2506214	11
Kovter	1/1 (100%)	160	3
Locky	55/55 (100%)	5672894	23940
NotPetya	1/1 (100%)	92	2
Ransomlock	1/1 (100%)	2312373	12
Razy	2/2 (100%)	3955	2851
SageCrypt	4/7 (57%)	3417095	9
Scatter	6/6 (100%)	5626959	560
Shade	2/2 (100%)	2900347	12613
Teslacrypt	82/82 (100%)	4351264	14
Torrentlocker	1/1 (100%)	2642555	388
Troldesh	2/2 (100%)	3500127	11
WannaCry	2/ 2 (100%)	5615288	162
Yakes	23/39 (59%)	2450372	9
Zerber	115/115 (100%)	5542697	70
Total:	495/524 (94%)		

That said, we need to comment that UShallNotPass was implemented before the Bad Rabbit and NotPetya ransomware families emerged. Therefore, until proven otherwise, we have at least one evidence that supports R2 that UShallNotPass can be effective on zero-day ransomware.

Case Study: *NotPetya*

We find it remarkable that UShallNotPass was effective against `NotPetya`, a particular debilitating ransomware that in 2017 was used for a global cyberattack primarily targeting Ukraine [8]. `NotPetya` is a ransomware which encrypts victim's disk at boot time (`NotPetya` has other malware characteristics such as the propagation, exploitation and network behaviors, but those are out of the scope of this thesis). Upon execution, `NotPetya` generates secure random numbers to use in the encryption, modifies the MBR of the system disk which enables it to load its own kernel in the next reboot. Next, it forces to restart the system and shows a fake `chkdsk` screen to the user. Meanwhile, the malicious kernel encrypts the MFT section of the disk which renders the data on that disk unusable. Since `NotPetya` loads its own kernel, the solutions proposed by [23, 25, 26] are bypassed and therefore cannot protect the victim. Moreover, [30] logs the random numbers that `NotPetya` uses to derive the encryption keys. Nonetheless, the key vault becomes inaccessible as well as other data after the reboot as the MFT is encrypted. On the other hand, UShallNotPass stops `NotPetya` once it calls `CryptGenRandom` and terminates it before any cryptographic damage occurs.

$870,000,000
Pharmaceutical company Merck
$400,000,000
Delivery company FedEx
$384,000,000
Construction company Saint-Gobain
$300,000,000
Shipping company Maersk
$188,000,000
Snack company Mondelēz
$129,000,000
Manufacturer Reckitt Benckiser

$10 billion
Total damages from `NotPetya`, as estimated by the U.S. White House.

Figure 7.4: Approximate damages reported by some of the `NotPetya`'s biggest victims [8].

Performance

We measured the overhead of UShallNotPass on computing and storage resources to answer **Q3**. Our assessment focuses two points: (i) API level overhead, i.e., the extra time to access secure randomness, (ii) application level overhead, namely, the latency perceived by the users. We conducted the assessments on a Windows 7 OS running on a VM with 2 CPU cores clocked at 2.7 GHz.

Benchmarks in API Level

We measured the time cost of invoking the `CryptGenRandom` API on the clean machine. For this aim, we wrote a benchmark program that invokes `CryptGenRandom` to generate 128 bits of random number, repetitively for 100 000[8] times and outputs the total time spent for this action. We observed that it took 0.12 seconds to complete this task. Then we run the benchmark program on the system that UShallNotPass runs. This time it took 15.59 seconds to complete the same task. The results states that UShallNotPass introduces an overhead with a factor of 125. According to our analysis, the main reason behind this impact is the significantly slow communication between Interceptor and Controller components of UShallNotPass. We also observed that, if the overhead of communication is discarded, the performance impact happens to be a factor of 5.52. We remark that the observations are made on an unoptimized prototype of UShallNotPass. More efficient techniques of IPC and dynamic decision making for access control would result in better performance figures.

8: We have chosen to set the limit of trials to 100 000 as with the current implementation of Inter-Process Communication (IPC), our setup becomes instable beyond this limit.

Our measurements on API level overhead and detailed results are illustrated in Table 7.2. It should be also noted that as the length of the pseudo-random number increases, the cost ratio of access control gets lower.

Impact in Application Level

Another important performance criterion is the slowdown in functionality of the software due to UShallNotPass. On our test system, we installed latest versions of select applications which are common in home and office

Table 7.2: UShallNotPass's performance impact on 100 000 iterative calls to `CryptGenRandom`.

Measurement Mode	Random Number Length (bits)			
	128	256	1024	2048
UShallNotPass Off (seconds)	0.12	0.15	0.20	0.27
UShallNotPass On (seconds)	15.59	15.80	15.84	16.91
Time spent in IPC (seconds)	14.90	15.05	15.05	16.00
IPC Discarded (seconds)	0.69	0.75	0.79	0.91
Total Overhead (factor)	125.42	105.68	77.69	61.77
IPC Discarded Overhead (factor)	5.52	5.00	3.89	3.32

users. Next, we whitelisted and run the applications while UShallNotPass is active. We inspected whether any slowdown occurred during the use of each application and logged the CSPRNG consumption, if any. The test set contains the following applications: 7zip, Acrobat Reader, Chrome, Dropbox, Firefox, Foxit Reader, Google Drive, Internet Explorer, LibreOffice, Microsoft Office, Putty, PyCharm, Skype, Slack, Spotify, Teamviewer, Telegram Desktop, TeXstudio, Visual Studio, VLC, WinRar and WinZip. Among those that called `CryptGenRandom`, we present our observations on the following five:

- **Acrobat Reader.** We created a new digital signature and signed a PDF document. During this period, Acrobat Reader called `CryptGenRandom` 13 times and obtained 64 bytes of random value in total.
- **Chrome.** We observed Chrome's CSPRNG usage by connecting a website over HTTPS. For this purpose, we connected `https://www.iacr.org/`. Once the TLS connection is established, we stopped monitoring. We recorded 2 calls to `CryptGenRandom` and 32 bytes of usage in total.
- **Dropbox.** After creating a new account, we put 5 files with various sizes, 20 MB in total. During the synchronization of these files, Dropbox invoked `CryptGenRandom` 61 times, obtaining 16 bytes of data in each.
- **Skype.** We monitored Skype when making a video call for 60 seconds. During this period, Skype performed 13 calls to `CryptGenRandom` and obtained 16 bytes in each call.
- **Teamviewer.** Among the tested applications, Teamviewer was the clear winner in pseudo-random number consumption. In our test, we connected to a remote computer and keep the connection open for 60 seconds. We observed 128 calls to `CryptGenRandom` which yield 2596 bytes in total.

In our tests, we did not notice any slowdown or loss in the functionality of any applications nor a program instability.

7.7 Discussion

History suggests that malware mitigation is a never ending race: a new defense system is responded with new attacks. We are no exception; cyber-criminals will attempt to develop new techniques to bypass our system. In this section, we first discuss how they could achieve this goal due to the limitations of our approach. Next, we review the issues may arise during the use of UShallNotPass.

Alternative Randomness Sources

The results of our experiments suggests that cryptographic ransomware can be efficiently mitigated by preventing access to CSPRNG APIs of the host system. Ransomware authors will try to find alternative sources for randomness. We anticipate that the first place to look for would be the *files* of victims. Generating encryption keys from files is known as *convergent encryption* [62] and already a common practice in cloud computing. That said, the feasibility and security of maintaining a ransomware campaign (from point of cybercriminals) based on this approach needs to be studied.

Alternatively, ransomware authors may try to fetch cryptographically secure random numbers (or encryption keys) from C&C servers instead of requesting access to CSPRNG APIs. As we discussed in Section 7.2 ransomware cannot establish a secure channel with the remote server in this scenario. Such a ransomware may still communicate with a randomness source on the Internet, over an insecure channel. In this case, however, the random numbers would be exposed to the risk of being obtained by IPSs. This would make it difficult for a ransomware to be successful in the long term. Having said that, more feasible defense strategies should be developed for home users who will likely not be in the possession of advanced network devices like an IPS.

Lastly, ransomware may statically link a random number generator and use a seed gathered from user space. However, this approach would require higher implementation effort and be error-prone. Furthermore, attempting to monitor events, e.g., keystrokes, would trigger the traditional antivirus software. Again, feasibility and security of this risky approach should be studied.

Implementation Related Issues

DLL Injection Method AppInit DLLs mechanism loads the DLL modules specified by the value `AppInit_DLLs` [9] in the Windows Registry. For ease of development, we utilized AppInit DLLs technique to load Interceptor component of UShallNotPass into target processes. However, in this methods, the payload DLL is injected by using the `LoadLibrary` function during the `DLL_PROCESS_ATTACH` phase of `User32.dll`. Therefore, executables that do not link with `User32.dll` do not load the AppInit DLLs [138]. Concordantly, UShallNotPass cannot intercept and control any calls made from these executables. We highlight that this limitation only concerns our current prototype, i.e., it is not inherent to the approach.

9: The value is stored under the key `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows` in registry.

Whitelisting Built-in Applications Modern OSes are installed with components including administrative tools and system utilities. Depending on the nature of the tasks, certain built-in applications may utilize the CSPRNG APIs. To keep the OS stable and secure, and maintain its functionality, these applications should be whitelisted before UShallNotPass launched. To determine which built-in Windows applications call CSPRNG APIs, we performed a clean install of Windows 7 32-bit on a VM, monitored the calls to CSPRNG APIs and identified the caller processes. During this experiment, we executed typical maintenance operations on the clean system, such as defragging hard disks, managing backups, installing drivers and updating the OS.

We detected invocation of CSPRNG APIs by Explorer (`explorer.exe`) and Control Panel (`control.exe`) which are two of the most frequently used Windows applications. Moreover, Windows Update (`wuauclt.exe`) and Windows Update Setup (`WuSetupV.exe`) are the only signed applications that consumed secure randomness. Therefore, if UShallNotPass is configured to allow the signed applications to access CSPRNG APIs, these two applications do not need to be whitelisted. Furthermore, Local Security Authority Process (`lsass.exe`) was the only application which calls `BCryptGenRandom`, while others called `CryptGenRandom`. The complete list of applications[10] that called CSPRNG APIs during the experiment is given in Table 7.3.

10: The list of applications may vary on different versions of Windows OS.

Table 7.3: Windows applications that calls CSPRNG APIs. Most of the applications listed below are located at `%WINDIR%\System32`. We remark that, only the executable files `wuauclt.exe` and `WuSetupV.exe` are digitally signed.

Executable Name	File Description
`explorer.exe`	Windows Explorer
`lsass.exe`	Local Security Authority Process
`SearchIndexer.exe`	Microsoft Windows Search Indexer
`svchost.exe`	Host Process for Windows Services
`dllhost.exe`	COM Surrogate
`wmiprvse.exe`	WMI Provider Host
`SearchFilterHost.exe`	Microsoft Windows Search Filter Host
`SearchProtocolHost.exe`	Microsoft Windows Search Protocol Host
`control.exe`	Windows Control Panel
`TrustedInstaller.exe`	Windows Modules Installer
`VSSVC.exe`	Microsoft Volume Shadow Copy Service
`WMIADAP.EXE`	WMI Reverse Performance Adapter Maintenance Utility
`wuauclt.exe`	Windows Update
`WuSetupV.exe`	Windows Update Setup
`mmc.exe`	Microsoft Management Console
`MpCmdRun.exe`	Microsoft Malware Protection Command Line Utility
`dfrgui.exe`	Microsoft Disk Defragmenter

Handling Sofware Updates Components of OS or user software may be updated for various reasons, including patching security vulnerabilities, fixing bugs and adding new functionalities. The update process may also involve replacing the existing executables with newer ones and thus altering their hash values. Therefore, if an OS component or an installed application which has access rights to CSPRNG APIs is updated, Whitelist of UShallNotPass must also be updated accordingly to prevent false positives. More precisely, the old hash value should be removed from the Whitelist and the new hash value should be added.

Abuse of Digital Signatures While Code Signing aims to help verifying the software origin, cyber criminals frequently used stolen certificates to sign malware in order to penetrate this defense [139, 140]. Furthermore, there is an incidence i.e., a ransomware sample with a valid digital signature [141], which proves that ransomware authors also have this capability. Such a clandestine ransomware sample may evade access control feature promised by our system. Namely, if UShallNotPass is configured to allow digitally signed applications to access CSPRNG of the host system, and the ransomware binary has a valid signature (e.g., the stolen certificate is not revoked yet or Certificate Revocation List (CRL) is not up to date), then the victim's files would be encrypted. Note that utilization of digital signatures is optional and meant to improve practicality and applicability of our system. System administrators should decide enabling this feature according to their systems' needs and capabilities. When ultimate security is desired, this option should be left as disabled so that even digitally signed ransomware would not cause harm on data.

Dynamic Decision As we discussed above, software applications on host system may be updated or replaced with another one. To prevent interruption of the work flow, UShallNotPass may be configured to ask administrator's permission in case a previously unseen process requests access to CSPRNG of the host system. This brings the risk of infection, as the administrator may not concentrate well each time. We remark that Dynamic Decision is an optional feature of UShallNotPass and is an example of security/usability trade off. If disabled, it would not pose any risk against security.

Efficient End-Point Protection from Ransomware

8

The core idea of UShallNotPass is to prevent ransomware from accessing to CSPRNG APIs. These functions offered by the OS return the essential ingredients required to build cryptographically secure encryption keys: "good" pseudo-random numbers. The solution described in Chapter 7 has a sufficiently accurate detection rate (i.e., 94%), but it is not yet an effective and efficient solution. What it needs is an access control system that guarantees at least three important requirements: (1) to rely on architectural components that are not vulnerable against known or arguable targeted attacks; (2) to have lower false positive rate; (3) to impose a negligible performance overhead.

In this chapter, we discuss improvements to the solution proposed in Chapter 7 that satisfies requirements (1)–(3). It meets (1) by avoiding IPC, a choice that is potentially vulnerable to named pipes hijacking (see Section 8.2). It meets (2) by bootstrapping and maintaining a *Whitelist DB* of honest applications that also call CSPRNG (see refsecbootstrapping). It meets (3) by showing that, when run in respect to vanilla system, our implementation has a negligible overhead (Section 8.3) over applications that use CSPRNGs, with a relative improvement of roughly two orders of magnitude with respect the prototype presented in Chapter 7. We also test the new implementation against a new set of 747 active real-world ransomware samples, and measure the false negative rate to confirm our strategy. we call our new prototype NoCry, in antithesis to the infamous `WannaCry`.

8.1 Security Assumptions

NoCry works under two assumptions, which is inherited from its predecessor UShallNotPass (Chapter 7): (i) at the moment in which the anti-ransomware is installed on a target system and before it becomes active and operational, the system is non-compromised; (ii) the host machine can run anti-virus software to detect, stop and neutralize common malicious actions such as keystroke logging, process injection, etc.

We also stress one key point once more. The original concept, and thus NoCry, has been conceived to work against cryptographically strong ransomware *only*. At least in the ransomware samples that we have analyzed, those are the ransomware programs that access secure random number sources. NoCry does not stop ransomware that does not follow secure development standards and, for instance, derives keys from a *non-cryptographic* PRNGs, like `rand` function in `C` runtime library or `System.Random` class provided by .NET framework. In §8.3, we argue that such ransomware variants are weak, cannot achieve success in the long term, or can be stopped otherwise. Therefore, NoCry is not all-in-one defense but meant to work side-by-side with (or even, integrated into) traditional anti-malware solutions or in combination with other anti-ransomware systems.

8.2 NoCry: Enhanced Protection

We believe that an anti-ransomware application should be effective and non-invasive in at least the following meanings:

1. **Robust Architecture.** Execution and operation of a defense system should rely on architectural choices that minimize the attack surface and have no vulnerabilities against known and arguable targeted attacks. In our case, the authorization mechanism need be robust against targeted attacks.
2. **Low False Positive Rate and Minimal User Intervention.** A defense system must provide security, and at the same time, must ensure (arguably and measurably) a low rate of false positive. The challenge regards our Whitelist DB. The list needs to be safely bootstrapped, and software updates should be reflected in the Whitelist DB with no interruption, inconsistency, or possibility of intrusions.
3. **Optimized Decision Procedure.** Performance impact of running an anti-ransomware should be negligible and must be imperceptible by the user. In NoCry, the overhead is due to the interception of calls to CSPRNG APIs and the time required by the access control decision procedure.

We discuss NoCry in the reminder of the section. We refer to Windows systems, as they have been the target of most of the ransomware attacks known at today. While we may refer to terms available particularly on Windows OS, what we discuss applies to other modern platforms as well.

Robust Architecture

As described in Chapter 7, UShallNotPass consists of two components: Interceptor detects the calls made to CSPRNG APIs, and Controller makes authorization decisions for the caller processes. This architecture needs an active communication channel between Interceptor and Controller components. In order to fulfill this need, UShallNotPass employs *named pipes*.

A named pipe is an IPC mechanism which enables processes to communicate to each other using a client-server architecture [142]. In this model, the *pipe server* is the application which creates the named pipe. Once the pipe is created, *pipe clients* – the applications that connects to the pipe server – can start sending/receiving messages to/from the pipe server. In the access control system of UShallNotPass, Interceptor creates two simplex named pipes, one for dispatching the process ID to Controller and another for getting the authorization result.

That said, named pipes in Windows platform are infamous with their security issues [143]. Among them, one particular issue constitutes a critical vulnerability for UShallNotPass. Namely, a malicious application can attempt to create a named pipe before the legitimate application does, and act like the pipe server. The pipe name of UShallNotPass is static and therefore a ransomware can hijack the pipe by creating the pipe instance more quickly than Controller of UShallNotPass. This would make the attacker owner of the named pipe object, allowing the ransomware to impersonate the Controller and authorize itself.

Observing this vulnerability, NoCry is designed to be IPC-free. In this new architecture, Interceptor and Controller are moved into Unified Agent, a single module which intercepts and controls CSPRNG calls. The architectural view of NoCry is illustrated in Figure 8.1. The capability of direct data exchange between Interceptor and Controller renders NoCry immune to the potential targeted attacks. Consequently, we conclude that NoCry is a more robust protection system.

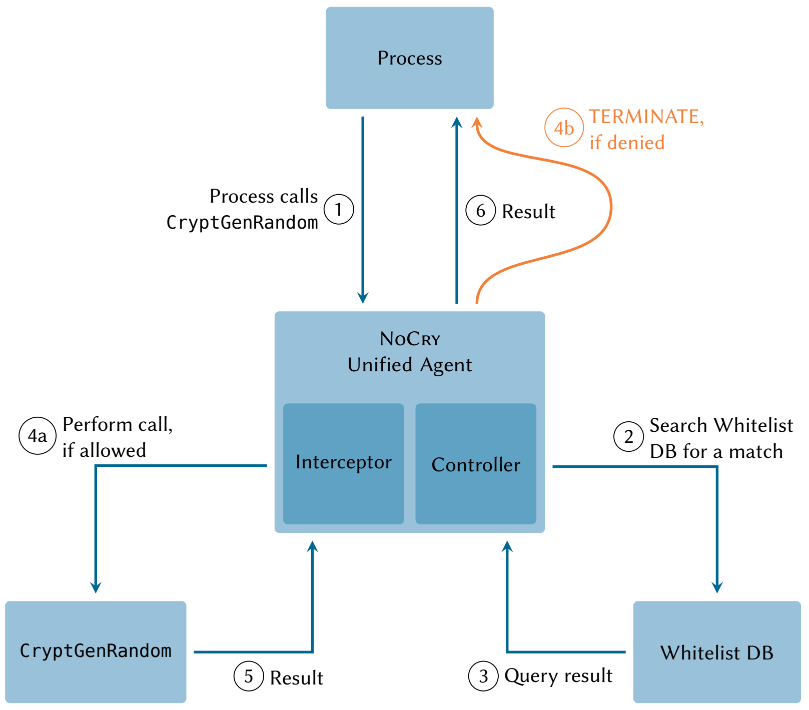


Figure 8.1: Architectural view of NoCry. Interceptor and Controller reside in the same module, Unified Agent. This new construction enables robust and efficient information exchange between Interceptor and Controller for making an authorization decision.

Low False Positive Rate and Minimal User Intervention

We introduce two methods that NoCry offers in order to increase the usability.

Bootstrapping Whitelist DB

UShallNotPass does not come with a pre-determined whitelist of benign applications. The list, presumably, is initially empty and if access control over CSPRNG APIs were applied immediately after UShallNotPass is installed, every cryptographic application invoking these functions would be stopped: this is surely not what the we mean to happen. Thus, benign cryptographic applications should be whitelisted before UShallNotPass is launched. To make this task as much automatic as possible we suggest in NoCry a *Training Mode*. It starts immediately after installation: the Interceptor listens the calls made to CSPRNG APIs without blocking any.

Under our assumptions (Section 8.1), all access requests to CSPRNG APIs should come from honest processes. The hash of the binary executables are added to Whitelist DB. Training Mode can only be activated once and just after the setup.

What if, against our assumption, Training Mode is run on a system that is infected by some strains of silent ransomware [144]? Such strains in fact infect computers but stay inactive until being activated by C&C servers or simply await until a certain time has passed. This way, ransomware attempts to look like a benign application and evade behavioral analysis-based detection systems. It is unlikely that such ransomware bypass NoCry: the ransomware executable would not call CSPRNG APIs in the sleeping phase and therefore they will not be whitelisted, unless the training phase coincides with the awakening of ransomware. This may be a remote possibility, but raises our assumption of making mandatory running our Training Mode in a clean system a must, as it is usually the case for any anti-malware.

Handling Software Updates

Whitelist DB can change. Programs that access CSPRNG APIs but are installed after the Training Mode has ended, must have their hashes be added to it. OS components are updated for various reasons, including patching security vulnerabilities, fixing bugs and adding new functionalities and since the update process involves replacing the existing executables with new ones, their hash values in Whitelist DB have to be updated consequently. User applications also regularly check for new updates and install them in the background. The hashes of these updated executables should also be reflected to Whitelist DB.

In environments where this could potentially lead to delays, e.g., due to slow human reaction, we suggest that NoCry can be configured to defer access control to keep the system stable and workflow uninterrupted. We call this *Deferred Mode*.

When working in Deferred Mode, NoCry does not immediately block calls to CSPRNG APIs coming from unknown processes. Instead, the parameters and outputs of these calls are securely logged in a protected location until administrator takes an action. Here, administrator can find the software benign, thus add the hash of the executable to Whitelist DB and dispose the logs associated with that process. Otherwise, the process is suspended and, if necessary, recovery procedure is initiated. The logging, and when necessary, recovery procedures are similar to the approach of PayBreak [30] which we discuss in Section 1.2. However, there are two notable differences in NoCry:

(i) logging is applied per unidentified process, not system-wide; and
(ii) once the administrator makes a positive decision, the logs are disposed.

The rationale of the variations above is to reduce the potential impact of logging the outputs of CSPRNGs. In our approach, random numbers obtained by whitelisted processes are not logged. This eliminates the security risks which could arise due to the persistence of the generated random numbers which are potentially used for cryptographic purposes.

Optimized Decision Procedure

In UShallNotPass, the access control over CSPRNG APIs requires to make an authorization decision which cause a significant delay. Mainly, the delay is due to two factors: (i) time spent for establishing IPC; and (ii) time spent by Controller for authorization.

As discussed in Section 7, the IPC is the main bottleneck of the authorization procedure and causes an overhead on CSPRNG APIs calls with a factor ranging from 62 to 125. In addition to the improved the security, eliminating the IPC from access control system is another motive which led us to unify Controller and Interceptor in a new module Unified Agent in NoCry. This way, both interception and authorization tasks are carried out in one place, without needing to consume time for IPC which enables to decide and act faster.

Furthermore, in UShallNotPass, the subsequent calls from the same process are authorized independently. While this approach would provide the highest level of time-granularity in access control, it might be an overkill for the security goals and a waste of resources for many systems. As we report in Section 7, the security checks performed in Controller causes an overhead up to a factor of 5.52. NoCry, therefore, holds an authorization to be valid for the lifetime of a process.

It is reasonable to expect that the two optimizations above would bring a significant performance improvement, which we assess in the next section.

8.3 Methods, Experiments and Results

On NoCry, we have run a series of experiments aiming at to measure the performance overhead, and *false positive & false negative* rates. For each experiment we describe the methodology, then we report and discuss the result.

Performance

Methodology. We measure the time that a benchmark program spends invoking `CryptGenRandom` API repetitively for 100 000 times. We run the benchmark program first on a clean system, then on a system with NoCry. We made this experiment on Windows 7 32-bit OS, running on a VM with 2 CPU cores clocked at 2.7 GHz. Overall, this is the same setting used in Section 7.6.

Results and Discussion. Table 7.2 shows the results of our measurements. It also reports the result from subsection 7, obtained using the exact same methodology.

Our analysis shows that NoCry brings drastically lower overhead in terms of time for getting the output of `CryptGenRandom` API. This improvement is due to the unification of Interceptor and Controller components of UShallNotPass which enables interception and control actions to be

Table 8.1: Time benchmarks of 100 000 iterative calls to `CryptGenRandom` API. Performance gain is calculated as $(old - new) \setminus old \times 100$. Measurements of UShallNotPass are recalculated.

Measurement Mode	Random Number Length (bits)			
	128	256	1024	2048
Clean System (sec)	0.13	0.14	0.18	0.24
UShallNotPass (sec)	15.59	15.80	15.84	16.91
UShallNotPass Overhead	11992%	11285%	8800%	7024%
NoCry (sec)	0.17	0.18	0.22	0.29
NoCry Overhead	30%	22%	18%	20%
Performance Gain	98.9×	98.9×	98.6×	98.3×

managed by a single component, Unified Agent, and thereby removing IPC. This result is not surprising after our improvements in Section 8.2 and confirms our hypothesis.

Another cause of the performance increase is the use of cache mechanism during authorization. In UShallNotPass, iterative calls from the same process are authorized individually, causing a significant overhead, as much as a factor of 5.52 (see Section 7.6). With NoCry, process authorization is valid for the lifetime of a process. That is, accessing to Whitelist DB is performed once after the first invocation of a CSPRNG API. This allows eliminating the need for accessing Whitelist DB for authorizing subsequent calls.

Lastly, the architecture of UShallNotPass limited the maximum number of iterative calls to `CryptGenRandom` API to the order of 100 000 as the system becomes unstable beyond this point (see Section 7.6). Since NoCry is IPC-free, it was able to handle a significantly larger number of requests. This makes it a better candidate for a protection system where CSPRNGs are heavily consumed.

Evaluation of False Positives

In the domain of NoCry, *false positive* describes the condition that a legitimate process calls a CSPRNG API and is stopped by NoCry.

Methodology. We have collected the Top 20 Installed Programs according to Avast PC Trends Report 2019 [145], and we look at whether they utilize CSPRNG functions *and* have digital signatures, the criterion which NoCry can use for authorization.

Results and Discussion. Table 8.2 presents the results of our findings. All of the applications in the Top 20 list calls CSPRNG APIs when running. Among the Top 20, the only unsigned application is `7-Zip`. Being `7-Zip` is an open source software, system administrators can study the code and add it to the NoCry whitelist if they find it safe.

It is reasonable to expect that digital signatures of applications and source code availability of open source software together help system administrators maintain Whitelist DB and therefore lower the number of false positives. In the lights of these circumstances, we perceive that the false positive rate of NoCry will be at a non-invasive level.

Rank	Program	Calls CSPRNG APIs	Digitally Signed	Open Source
1	Google Chrome	✓	✓	
2	Acrobat Reader	✓	✓	
3	WinRAR	✓	✓	
4	MS Office	✓	✓	
5	Mozilla Firefox	✓	✓	✓
6	VLC Media Player	✓	✓	✓
7	Skype	✓	✓	
8	CCleaner	✓	✓	
9	iTunes	✓	✓	
10	TeamViewer	✓	✓	
11	Windows Live Essentials	✓	✓	
12	7-Zip	✓		✓
13	Stream	✓	✓	
14	Dropbox	✓	✓	
15	Opera	✓	✓	
16	CyberLink PowerDVD	✓	✓	
17	CyberLink PowerDirector	✓	✓	
18	HP Photo Creations	✓	✓	
19	CyberLink YouCam	✓	✓	
20	CyberLink Power2Go	✓	✓	

Table 8.2: Top 20 Installed Programs according to [145]. All applications in the table calls one or more CSPRNG APIs. NoCry will allow these calls automatically since the applications are digitally signed, except for 7-Zip, which is an open source software.

Evaluation of False Negatives

The analyses in previous works [13, 45] recognizes the following three strategies to obtain the encryption keys: (i) using embedded keys in the binary file; (ii) generating keys on the victim's machine; and (iii) downloading keys from a certain network location. The security analyses of key generation in ransomware are found in [13, 45]. Here, we resume it. If a ransomware follows (i), keys can be extracted from the ransomware binary, and the encrypted files can be recovered. Most of the ransomware prefer to generate the keys on victim's machine. In this case, there are two options: to use the CSPRNG, which produces high entropy random values; or to use a non-cryptographic PRNG. The first has been largely discussed already. The second is a weak choice: PRNGs are designed to be reproducible thus their outputs are guessable. If the ransomware uses a non-cryptographic PRNG, like `rand` function in `C` runtime library or `System.Random` class provided by .NET framework, decryption is feasible. If ransomware fetches keys from a remote server (iii) then blocking the malicious IPs inhibits the ransomware, which forces ransomware developers to fallback to (i) or (ii). The only option for current ransomware to get good encryption keys is therefore to use a CSPRNG.

In order to support our argument that NoCry can stop ransomware from accessing CSPRNG, and thereby prevent its damages, we designed an experiment. The outcome also measures the false negative rate of NoCry. Next, we check if there exists a publicly available decryptor for those samples that did not call any CSPRNG APIs.

Methodology. Following the same methodology in Section 7.6, we (1) obtain a fresh malware corpus from VirusTotal[1]; (2) pick potential ransomware among them; (3) rebuilt the same test environment, using Cuckoo Sandbox[2] to identify the active ransomware samples; and, (4) classify the families using AVclass [128] tool; (5) run NoCry against them; and (6) (if exists) discover the reason for a false negative.

1: VirusTotal Threat Intelligence, `https://virustotal.com`

2: Cuckoo Sandbox, `https://cuckoosandbox.org/`

Table 8.3: List of active ransomware samples tested against NoCry. The notation x/y means that *x samples out of y could be successfully stopped.*

Family	Samples (%)
Barys	1/1 (100%)
Birele	1/1 (100%)
Bitman	152/152 (100%)
Browserio	2/2 (100%)
Bzub	1/1 (100%)
Carberp	0/1 (0%)
Cerber	60/60 (100%)
Cryakl	0/1 (0%)
Cryptxxx	0/2 (0%)
Crysis	2/3 (66%)
Dalexis	1/3 (33%)
Daws	5/5 (100%)
Delete	1/1 (100%)
Deshacop	1/1 (100%)
Dlhelper	1/1 (100%)
Enestaller	1/1 (100%)
Enestedel	1/1 (100%)
Expiro	1/1 (100%)
Gamarue	2/2 (100%)
GandCrab	1/1 (100%)
Gator	1/2 (50%)
GlobeImposter	1/1 (100%)
Godzilla	1/1 (100%)
Jaff	1/1 (100%)
Lethic	4/4 (100%)
Locky	47/47 (100%)
Midie	1/1 (100%)
Neoreklami	0/1 (0%)
Occamy	4/4 (100%)
OpenCandy	2/2 (100%)
Petya	2/2 (100%)
QQPass	1/1 (100%)
Razy	6/6 (100%)
SageCrypt	1/1 (100%)
Saturn	1/1 (100%)
Scar	3/3 (100%)
Scatter	2/2 (100%)
Shade	2/2 (100%)
ShadowBrokers	1/1 (100%)
Shiz	17/17 (100%)
Sigma	0/1 (0%)
Sivis	3/7 (42%)
Spigot	2/2 (100%)
Spora	2/2 (100%)
Striked	0/1 (0%)
Swisyn	0/1 (0%)
Tescrypt	5/5 (100%)
TeslaCrypt	316/316 (100%)
Tpyn	1/1 (100%)
Upatre	2/7 (28%)
Ursnif	1/1 (100%)
Vobfus	1/1 (100%)
Wowlik	1/1 (100%)
Wyhymyz	1/1 (100%)
Zerber	52/52 (100%)
Zusy	7/7 (100%)
Total:	726/747 (97.1%)

Results and Discussion. We identified a new set of 747 active samples from 56 cryptographic ransomware families. Next, we installed NoCry on the test machines and run the executables against NoCry. Table 8.3 shows the results: 97.1% of the samples have been stopped by NoCry before any user file is damaged, i.e., encrypted by the ransomware program. They were the samples that attempted to call CSPRNG during the attacks, and were terminated by NoCry as they were not present in Whitelist DB.

Among the 2.9% of samples that cause false negative, there may be ransomware executables that either circumvented NoCry's access control, or ransomware process did not call CSPRNG APIs. To discover the exact reason behind the false negatives, we picked random samples from the families we missed, and manually analyzed the API call tree. The missing samples from `Cryptxxx` and `Dalexis` did call `CryptGenRandom` API, however, said API could not be hooked by NoCry. We believe this is due to a problem of our implementation. The missing samples from `Carberp`, `Cryakl`, `Crysis`, `Gator`, `Neoreklami` and `Sigma` families did not call any CSPRNG APIs. Among them, we found decryptors for `Cryakl`, `Crysis` and `Sigma` on ID Ransomware[3] platform.

3: ID Ransomware, https://id-ransomware.malwarehunterteam.com/.

8.4 Discussion

The solution we proposed to address the second issue, i.e., how to manage the Whitelist DB, needs further discussion. In a system assumed uncorrupted, we bootstrap the list in *Training mode* by feeding in honest applications that call CSPRNG. In *Deferred mode* we update the list when a new version of a whitelisted application is available; we temporarily grant it the right to call CSPRNG but retaining critical data that can help recovering files in rare case where the upgrade hides a ransomware. Despite looking reasonable to us, one can still challenge our choices. For instance, one can ask why managing a Whitelist DB of applications that call CSPRNGs in the first place? In fact Windows OS already offers a protection, `AppLocker`, that enables to deny non-whitelisted apps (e.g., malware) from running. Cannot be ransomware dismissed as any other malware? First, we observe, this practice seems not have slowed down ransomware so we conclude that it needs more time and maturity to be widely accepted. Second, the problem with the whitelists is that they may not be complete, generating fastidious false positives. This issue, of course, affects also NoCry, but differently from a system which offers protection against generic *harmful* apps (a term that may have different interpretation). NoCry targets and operate against a very specific situation. If we imagine to defer to the user the decision about whether a potential false positive is indeed so, NoCry can precisely state that a certain application is trying to call critical functions, potentially to create strong encryption keys and unless the application is meant to encrypt data, it is better to let NoCry kill it. We fail to imagine instead stating a similar precise claim to warn about a generic harmful application. The best could be a warning message sounding like "something insecure may happen", alert that users have learned to ignore [146]. A precise claim like that, enabled by NoCry, will help users take more informed decisions, arguably reducing the number of false positives, and we intend to test this hypothesis in a future work.

Another critic can be that by only guarding access to CSPRNG, we miss to stop ransomware that generate encryption keys using different strategies. If this critics were well founded, this would count as a serious deficiency for NoCry because would lead to false negatives. To this critic we answer first by observing that NoCry was neither designed to stop other ransomware than those calling CSPRNG APIs nor conceived to work in isolation from other anti-ransomware. Indeed, we think, the full potential of NoCry will emerge only in integration with an anti-malware that provably can reverse the damage done by ransomware that make use of the encryption keys obtained not by calling CSPRNG. Theoretical solutions exist that have the potential ability to reduce the cost of reversing encryption to a feasible time complexity and a few solutions are actually implemented [147]. These seems, indirectly, to support the argument that by not calling CSPRNG APIs, ransomware can realize only cryptographically-weak encryption. It is then by uniting different methods that we imagine a good anti-ransomware can reliably combat this crypto-crime. How to do it properly is an open problem but considering the improvements that we have herein discussed, we believe this union is possible and practical.

Future of Ransomware

Future of Ransomware 9

In this chapter we discuss what potential strategies could ransomware authors might develop to continue causing damage for an extended period: we introduce original ransomware variants that employ rootkit techniques and white-box cryptography, and, inspired by the cybersecurity incidents occurred in real-world applications, we point out new possible ransomware targets and attack types.

9.1 Potential New Threats

We start by giving high-level descriptions of advanced techniques that ransomware may utilize to defeat the defense systems analyzed in the previous chapters. Next, we point out new areas that ransomware may exploit and extend the attack surface that next generation ransomware may target. In each discussion, our observations are supported by the real world incidents.

Rootkit-based Ransomware

Rootkit is a type of malware that has the ability to conceal its activities on the target computer system, e.g., code executions, file I/O, network and connections [148]. The capability of hiding malicious operations is achieved by hooking operating system's APIs in order to filter and remove the rootkit's traces, as depicted in Figure 9.1. Since a rootkit clears its footprints from APIs that inspect file and memory access, the rootkits are harder to detect than other types of malware.

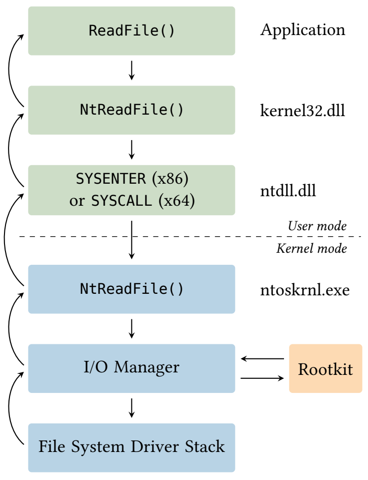


Figure 9.1: Interception of read calls by a rootkit in order to hide its trace.

Hooking system APIs can be accomplished in several ways, including changing the function addresses in IAT, patching SSDT in kernel level, and injecting code into applications (DLL injection) [149]. Starting from Windows Server 2003, x64-based versions of Windows platform introduced Kernel Patch Protection (KPP) which forces kernel mode drivers to be digitally signed, hence prevents unknown modification of code or critical structures in Windows kernel [150]. Nevertheless, cybercriminals frequently used stolen certificates to sign malware in order to penetrate this defense [139, 140]. Ransomware authors also seems to have this capability. A VirusTotal report shows that a sample of `Razy` ransomware has a valid digital signature [141].

Implementations of current ransomware defense approaches deeply rely on the security guarantees of the host OSs. While increasing the bar for cybercriminals, state-of-the-art ransomware defense systems utilizes user mode hooks or kernel mode drivers to monitor behavior of applications and stop ransomware [23, 25, 26, 30]. Although there is currently no known ransomware which utilizes the advanced techniques of rootkits, the aforementioned defense systems may not detect a rootkit-based ransomware.

Obfuscation

Listing 9.1: Assembly code of a computer program that multiplies 1 and 2.

```
push    rbp
mov     rbp, rsp
mov     WORD PTR [rbp-2], 1
mov     WORD PTR [rbp-4], 2
movzx   eax, WORD PTR [rbp-2]
movzx   edx, WORD PTR [rbp-4]
imul    eax, edx
mov     WORD PTR [rbp-6], ax
movsx   eax, WORD PTR [rbp-6]
pop     rbp
ret
```

Listing 9.2: Obfuscated version of the original program in Listing 9.1. Semantic equivalence is preserved — this program also multiplies the integers 1 and 2. The transformed function is obtained by adding ineffective instructions shown inside red boxes. Note that the code's appearance is changed while keeping its behavior same.

```
push    rbp
mov     rbp, rsp
mov     WORD PTR [rbp-2], 1
mov     WORD PTR [rbp-4], 2
movzx   eax, WORD PTR [rbp-2]
add     eax, 1
mov     WORD PTR [rbp-2], ax
and     WORD PTR [rbp-2], 32767
movzx   eax, WORD PTR [rbp-2]
sub     eax, 1
mov     WORD PTR [rbp-2], ax
and     WORD PTR [rbp-4], 32767
movzx   edx, WORD PTR [rbp-4]
movzx   eax, WORD PTR [rbp-2]
imul    eax, edx
mov     WORD PTR [rbp-6], ax
movsx   eax, WORD PTR [rbp-6]
pop     rbp
ret
```

Obfuscation is the practice of making a software implementation incomprehensible through a sequence of transformations while preserving the program semantics [151]. Originally, legitimate vendors utilized obfuscation to protect intellectual property in software implementation. However, malware authors also take advantage of obfuscation to conceal malicious executable code in the binary programs. Concordantly, obfuscated malware can evade from *signature based detection* techniques which is one of the oldest approaches in the battle with malware.

Obfuscating malware can be categorized into four types: *encrypting*, *oligomorphic*, *polymorphic* and *metamorphic* malware [152]. The members of the first type encrypts malicious code segment in the binary program and decrypt it in the runtime. This involves a decryptor function embedded in the malware body to decrypt and execute the malicious code. Anti-malware systems, though, would still recognize the decryptor function and identify malicious software. Thus, the second type, oligomorphic malware, carries a set of encrypted decryptors in data segment of binary and changes the decryptor in each generation. However, the number of decryptors is limited and therefore all of them eventually gets identified by anti-malware systems. On the other hand, polymorphic malware mutates its decryption engine randomly, hence evades signature based detection. The means of mutation include dead code insertion, register reassignment, subroutine reordering, instructor substitution, code transposition & integration. For instance, dead code insertion is the practice of adding code that has no effect on the functionality of the software and is shown in Listing 9.2. For the details of other techniques, we refer the reader to [153]. Anti-malware vendors developed *sandboxing* approach to help detection, which works by observing the program's behavior in a safe environment. Once the polymorphic malware is executed in sandbox and the constant malicious part is decrypted in the memory, signature based detection can be applied. The race between cybercriminals and anti-malware vendors resulted the appearance of metamorphic malware which actively recognizes, parses and mutates its whole body. As it does not contain a constant body, and thus cannot be detected via signature analysis [154], metamorphic malware has been considered to be the most dangerous type.

In the ransomware side, the situation seems to be safe for now. As of today, there is no known instance of obfuscated ransomware through aforementioned techniques. Contemporary ransomware utilizes binary packers, e.g., UPX[1], ASPack[2] or PEtite[3], which are used to compress the compiled code in order to make the size of executable even smaller. However, malware authors do not confine themselves to well-known packers, often write their own obfuscator routines and utilize combined packers [155]. This multi-layer protection may hinder defense systems based on API monitoring (if third party crypto libraries statically linked) and sandboxing. In the case of an unlucky event of infection, such a ransomware can be devastating.

1: Ultimate Packer for eXecutables, https://upx.github.io/

2: ASPack, http://www.aspack.com/aspack.html

3: PEtite, http://www.un4seen.com/petite/

White-Box Cryptography

White-box cryptography is the concept of protecting the sensitive data hard-coded in a software implementation [156, 157]. In particular, main focus of this domain is to embed secret keys into the source code in such a way that it is hard to extract them from compiled binary. An example of a Feistel network based block cipher and its fixed-key white-box implementation are illustrated in Figure 9.2 and Figure 9.3, respectively. Although white-box cryptography is not a new idea (it is first introduced in 2002), no secure white-box implementation of the block cipher AES exists yet, for instance, previous proposals are found to be open to key extraction and table-decomposition attacks [158]. Nevertheless, white-box cryptography still continues to be an active field of research [159–161].

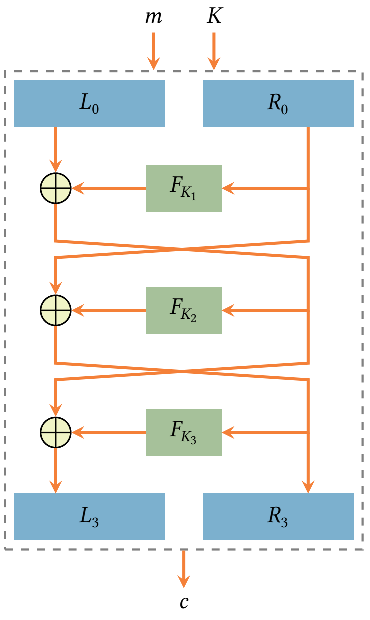


Figure 9.2: Illustration of a block cipher algorithm based on a 3-round Feistel network structure.

Currently, ransomware implementations cannot protect the secret keys in the memory during the encryption process. Using this weakness, defense systems can extract these keys using various techniques. For instance, a key escrow like approach monitors calls to known cryptographic APIs (either built in or third party) and stores parameters of encryption functions in a vault [30]. In virtual environments, point-in-time snapshot of memory would also reveal those keys and recovery could be possible. Furthermore, some ransomware families encrypt victim's files using a key which is hard-coded in the ransomware body [162]. In this case, binary analysis can be utilized to search for static encryption keys in the compiled code. In other words, one can interact with the ransomware and propose solutions if the encryption keys resides unprotected in the memory. That being said, key extraction from securely implemented white-box algorithms is meant to be hard. Therefore, introducing of secure white-box implementations of block ciphers can tip the balance in favor of ransomware authors.

Ransomware of Things

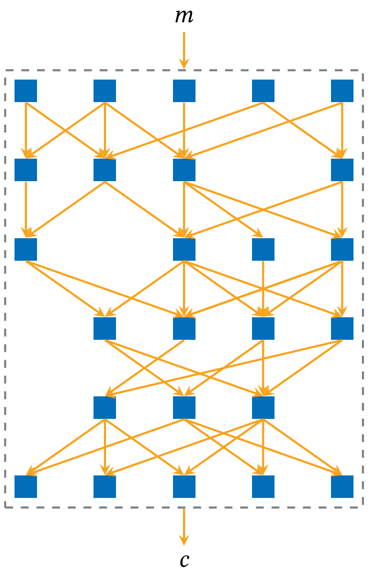


Figure 9.3: Illustration of a white-box implementation of that block cipher where the key K is hardcoded into the algorithm in a way that it is not possible to extract K.

Internet of Things (IoT) refers to the interconnected network of physical devices that can communicate over the Internet [163]. An IoT device can be equipped with electronic components, firmware, software, various types of sensors to collect information and actuators that allows to interact with the physical environment. Besides electronic devices like televisions, mobile phones and surveillance systems, in today's world, cars, planes, buildings, kitchen gadgets and even toys are also connected to the web.

IoT devices has been a part of our daily lives for a long time and can be seen virtually everywhere. However, IoT devices are inherently resource-constrained (CPU with low clock rate, small memory size). As such, the available options for cryptographic algorithms to use is limited when designing a secure communication protocol [164]. The security issues with IoT have always been a concern in information community [165], most importantly access control problems.

Given that the vulnerabilities in IoT devices and the high motivation of cyber criminals, there have already occurred several alarming and threatening ransomware incidents as follows. Hackers took control of ticket machines of San Fransisco's public transportation network and claimed ransom [166]. Furthermore in Austria, a hotel had to pay ransom after a ransomware infected its management system and blocked generating

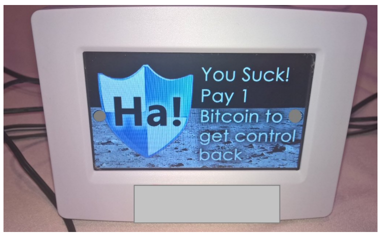


Figure 9.4: Vulnerable IoT devices, like this thermostat, constitute an easy target for ransomware. *Courtesy of Ken Munro.*

new cards [167]. Researchers demonstrated a proof-of-concept that the control of an Internet-enabled thermostat can be taken by a ransomware, allowing them to change the heating settings [168]. The screen of the hacked device is shown in Figure 9.4. Similarly, a security report states that cybercriminals launched a Permanent Denial of Service (PDoS) attack on IoT devices which wipes all data on the device and destroy its firmware and/or basic functions, causing a permanent corruption [169].

By extending the attack surface and lack of adequate security, IoT has a potential of opening doors to novel ransomware attacks. For example, researchers demonstrated that it is possible to take control of a car and remotely stop it [170]. Also, another group of researchers showed that 75% of bluetooth smart door locks can be wirelessly hacked [171]. Given these facts, it is reasonable to ask the following questions: Consider that your car was remotely stopped in a rural area. *Would you pay the ransom to re-activate the car's engine?* Likewise, when you return your home in the middle of the night and see that your door is locked. *Would you pay the ransom to go in your home?* The picture may become worse for the enterprises, as the ransom amounts can be set higher and this makes the enterprises a more plausible target for cyber criminals. But the negative effects of a ransomware attack is beyond the money: the damage in the reputation and work loss should also be counted. Taking into the account that the security flaws in IoT devices do not seem to be fixed soon, or even fixable [172], ransomware attacks may gravitate towards IoT in the near future.

Socio Technical Attacks

The ultimate goal of cyber-criminals is to obtain money as much as possible. To achieve this, they can become very creative and employ novel marketing strategies. In one of these, a ransomware variant called Popcorn Time offers an option to victims who want to get decryption keys without paying. The condition is first victim infects other two ones and these two victims pay the ransom. Then, the first victim obtains the keys. The initial samples of Popcorn Time ransomware have an encryption key embedded in the malware body [173]. Although the key can be extracted from the current sample of Popcorn Time and files can be recovered for now, previous evolution of ransomware suggests that future samples of Popcorn Time may become more effective.

To this day, the vast majority of famous ransomware families share the same principle. Extortion by holding decryption keys can be expected to succeed when its vital for victims to regain access to their data. However, on the other side of medallion, there is another fact. Some data may need to be kept private such that when leaked, data owner may lose advantage and/or have economical damage. Thus, another way to extort victims can be to exfiltrate sensitive data and ask for a ransom to not make it public. These data types may include trading secrets, financial records, medical history, government documents, details of high-tech projects, blue-prints of critical infrastructures, and internal/private communications. For example, the disclosure of data breaches reduced the purchase price of Yahoo by $350 million when it is acquired by Verizon [174]. It comes to mind that, instead of selling the leaked data in the underground market, hackers can try to claim a ransom to get a higher revenue. Another attack hit Sony Pictures, hackers compromised the computers and released sensitive data including company's financial records and e-mail messages of executives [175]. The contents of the breach put the company in a difficult situation so that one may ask the question: *Would Sony Pictures pay a ransom if attackers demand it?*

Lastly, we would like to point an important difference between extortion via encryption and data exfiltration. In the former case, the instance of threat comes to an end when the victims regain access to their files. In contrast, no one can guarantee that could retain cyber-criminals from asking for ransom again in the latter case. In this situation, it would be safe to expect that extortion via stealing sensitive information may be an increasing trend in the near future and prepare the network infrastructures against this threat.

9.2 Dual Use of Ransomware

In anti-ransomware research, ransomware samples are routinely analyzed. The goal is to understand how they generate or retrieve the encryption keys; how they search, sort and prioritize which files to target first; and which files they encrypt first and by using which encryption algorithm. In this quite methodical work, it is routine to reverse engineer ransomware samples and analyze their source codes. While performing this task, we found that some piece of code was not original but *copy-and-pasted* from well-known public repositories or developpers communities. From this discovery, with some additional work, we managed to build a decryptor for those ransomware samples.

Although our discovery is not surprising—researchers have already commented on how codes from public repositories is re-used and how this impacts security (e.g., see [176])—realizing that also ransomware's security depends on public code has captured our attention. We started wondering whether there were other cases of copy-and-pasted code in ransomware. And we started reflecting on which consequences such re-use of code may bring into the fight against ransomware attacks. This articles report on our insights on the subject.

Although motivated by some experimental findings, our contribution is purely argumentative. But, by developing our argument rigorously, we hope to contribute to a scientific discussions on "the matter". And being "the matter" related to *dual-use of concern* in ransomware research, we intend to embark on other questions as well: What famous precedents exist in the recent history of ransomware that could enlighten us on the pros and cons of dual-use research? Should ransomware be considered components of a cyber-weapon? And, as such, are there reasons to classify ransomware as having military use? Thus, would it be reasonable to resort to intelligence and counter-intelligence strategies, such as those suggesting to contain information spreading in case of an attack or to control public information, to mitigate the threat?

We restrict our argument to cryptographic ransomware, those which rely on cryptography. Other kind of ransomware, e.g., those which aim to distress victims to pay up but, like the *scareware*, only pretend to use encryption but do not, are excluded from the discussion.

Findings and Analysis

Is copy-and-paste from public repositories a practice in ransomware engineering? To investigate the question we have first to collect and obtain the code of real-world ransomware samples and reverse engineer it.

The most accurate way to accomplish this latter is to *decompile* the malicious binaries. The task becomes quite practical if the malware is implemented using the .NET framework. Looking into malicious .NET assemblies downloaded from "Hybrid Analysis", an automated malware analysis platform [177]. Hybrid Analysis utilize sandboxing technique to determine if an executable exhibits malicious behaviour or poses no specific threat. From it, we collected ransomware samples by searching on report database with the following settings: (i) Exact Filetype Description as `Mono/.Net assembly, for MS Windows`; (ii) Verdict field as `Malicious`; and (iii) Hashtag field as `#ransomware`.

On a initial set of 128 executable, we applied dnSpy [178], a tool to obtain source codes. 39 samples, obfuscated, precluded any analysis. Of the remaining 89, we manually perused the source code, searching for key generation and encryption routines. 68 samples turned out to be non-cryptographic ransomware, with no such routines in their program body. The remaining 21 cryptographic ransomware samples were our final data set.

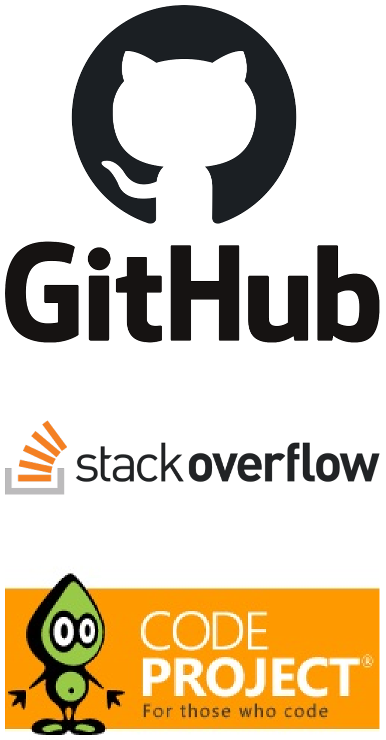


Figure 9.5: In our research, we investigated popular platforms among software developers: GitHub (`https://www.github.com`), StackOverflow (`https://www.stackoverflow.com`) and CodeProject (`https://www.codeproject.com`).

Using the found crypto-related code lines (e.g., key derivation, encryption) as keywords, we searched for those lines in source code repositories, question&answer platforms and developer communities (see Figure 9.5). When analyzing the hits, we compared the semantics of code snippets, naming of constants and variables, function signatures, strings, and error messages. From this searching and matching we discovered that some code was a *verbatim* copy-paste. Other code resulted, at least apparently, a plagiarism of some public available code.

Were we witnessing code-reuse (i.e., dual-use) in ransomware? Before claiming code-reuse, we had to verify whether the code had been published before the first appearance of it in the malware. There should also be a reasonable time frame between the two events. The date of the first appearance of a ransomware, checked by using VirusTotal [127], has been compared with the date on which the knowledge was first shared on online. A double-check on the integrity of the pieces of information available on the executable was also performed. According to our findings, at least 9 out of 21 ransomware samples resulted to contain snippets bearing a marked resemblance to codes at online resources, this leading us to conclude that they are in fact a copy-paste.

In the following, we can comment on an excerpt from the ransomware samples (see Table 9.1) that we have found being a copy-paste from (i) a public repository of fully functional ransomware prototypes; (ii) tutorials and posts at developer communities. We also elaborate, where possible,

Table 9.1: SHA256 digests and family names of the samples. To determine the family name, we applied AVClass tool on the labels provided by AV vendors which we obtained from VirusTotal. SINGLETON denotes that the tool could not find any plurality among the labels for that sample, i.e., no vendor agreed on the family name.

#	SHA256 Digest	Family	Reuse From
1	0e5a696773b0c9ac48310f2cda53b1742a121948df5bcb822f841d387f0f5f68	Jigsaw	
2	1d57564398057df99d73cca27015af24142c25828287837c73d2daf0b3c3af5b	Mimikatz	
3	1ebdbfea6ab13f258a7d00dea47de48261cfb84d52ebbb6f282498c3ab1b1b39	Occamy	
4	3b4aaf37510c0f255e238c81b7e1a446bfa925bd54f93969c3155d988fbb6501	HiddenTear	[179]
5	41ee4623d60544dd0ca16f6177565d99825afb38b932ccecc305ef2fc20e03f4	HiddenTear	[179]
6	58d11ef74b062e9996e75d238501a3f4d23691b101997d898d478696795ae3ff	CloudSword	[180]
7	662d0f034f2852e4e43d22a3625c1c8600c3d36660b596db1d6bad5c4980d9df	Ryzerlo	[179]
8	66a3172e0f46d4139cc554c5e2a3a5b6e2179c4a14aff7e788bb9cc98a2219d5	Tiggre	
9	7cdd7e30c7091fd2fa3e879dd70087517412a165bf14c4ea4fd354337f22c415	HiddenTear	[179]
10	87ce0b2e22b02572146676277cd6e9d89225e75361d1b696555cfe695c2e1f45	SINGLETON	
11	894aa842c129b39c0b9a7d575133d68b25de2ecd4e777f29e58481d30dfb6f4e	Omegax	
12	950be5b5501ee84b1641c3a9a780242a57cdd412892c781eac8781498bf11f3e	Bobik	
13	951d78dd92eba7daa3ef009ce08bba91a308e13bdeb8325af35bc8202bd76e9b	Tiggre	
14	b5f3a090556ea30210a23fc90b69c85c68e8e08c89fbe58eb6a829e356dcc42e	Occamy	
15	ce53233a435923a68a9ca6987f0d6333bb97d5a435b942d20944356ac29df598	Crypren	[181] & [182]
16	d36e6282363c0f9c05b7b04412d10249323d8b0000f2c25f96c6f9de207eedf8	HiddenTear	[179]
17	def09368d22c7b3f6a046ef206a57987095b2f4ddae1d26c6ef2594d6be09bfc	Diztakun	
18	e2ac9692c0816ccd59d1844048c6238dc5d105b0477620eeb1cdb0909804a787	WhiteRose	[179]
19	f37080ee4cc445919cae0b1eb40eff46571f7ce0d85b189321d80a41c8752212	SINGLETON	
20	f535879cf05a099bf0f6d2a7fa182d399ec9568f131abb23d9fb98418f45789d	Perseus	[180]
21	fd99bfeac78c087a9dc9d4c0c1d26a7ea9780a330f88ba0d803f3464221b4723	SINGLETON	

about where the original code comes from, and about its cryptographic qualities.

Ransomware from repositories of fully functional prototypes. Tiggre, see Table 9.1, is a sample of cryptographic ransomware that uses a key generation function that is copy of a piece of public code known as HiddenTear [179] (see Listing 9.3 and Listing 9.4). From it, Tiggre inherits a weakness: the password is generated using the outputs of a cryptographically weak algorithm. In fact, the same author of HiddenTear had developed a decryptor by using this weakness [183]. We tell the full story later, but what counts for now is that the open-source ransomware HiddenTear is a very famous ransomware code, which was posted publicly in 2015 allegedly for educational purpose. Since then, cyber-criminals have been using it as a source of inspiration for their ransomware variants [184]. This was also the case for Tiggre.

The original HiddenTear works as follows: it generates a password by calling CreatePassword which is shown in Listing 9.3. The password, from which the encryption keys are derived, is sent to C&C server. Next, before notifying the user, the ransomware attempts to encrypt all the files in test folder under the user's Desktop directory.

Ransomware authors that copy from HiddenTear had to implement their own back-ends before having a working ransomware, but HiddenTear remains their point of reference. We have found that basic functionalities such as password generation and encryption blocks have been replicated from HiddenTear: for each file, the encryption key is derived from the same master secret, the password; this latter is generated using System.Random, a class that provides (cryptographically weak) pseudo random numbers.

From a cryptographic point of view, the outputs of `System.Random` is reproducible when using the same seed and its secrets are vulnerable to a forensics analysis. But other variants of `HiddenTear` eliminate this weakness: the weak key generation method is not seen in those samples.

Listing 9.3: Password generation method of `HiddenTear`. This password will later be used as the master secret to derive encryption keys.

```
public string CreatePassword(int length) {
    const string valid = "a..zA..Z1234567890*!=&?&/";
    StringBuilder res = new StringBuilder();
    Random rnd = new Random();

    while (0 < length--)
        res.Append(valid[rnd.Next(valid.Length)]);

    return res.ToString();
}
```

Listing 9.4: Password generation method used by `Tiggre`. The set of valid characters is shortened, most probably, to ease the typing of the password when asked for recovery.

```
private static string RandomString(int length) {
    string chars =  "a..zA..Z0123456789";
    StringBuilder sb = new StringBuilder();
    Random random = new Random();

    while (0 < length--)
        sb.Append(chars[random.Next(chars.Length)]);

    return sb.ToString();
}
```

Ransomware from community platform. Confidentiality of data is a highly demanded and legitimate need in the digital world. While cryptographic techniques can be used to protect the secrecy of data, developing a security application is an error-prone process. Therefore, developers who recently entered in the field of cybersecurity might need to use the help of online tutorials. For example, Listing 9.5 shows a post on CodeProject website [181] which explains a simple way to encrypt a file using a key derived from a password in C# language. The function, `EncryptFile`, is poorly written from a cryptographic point of view. There are weaknesses, such as (i) presence of a hard-coded secret in the code; and (ii) improper key derivation, to name a few. That said, we found a `Crypren` ransomware variant (see Table 9.1) which copies the file encryption and decryption functions from [181]. Listing 9.6 shows the function modified by the ransomware author, who disdained to write a password generation method and even used almost the same hard-coded secret.

Furthermore, the same `Crypren` sample contains the exact code snippet shared at another developer community [182]. That piece of code, is meant to impersonate another user, i.e., to launch a process under that user's account. However, the said code portion is not used/referenced by the program.

In another case, we observed that an online tutorial published in 2005 inspired two ransomware samples: `Perseus` and `CloudSword` (see Table 9.1). The post, available at [180], explains how to encrypt files with a user-supplied password in VB.NET programming language. Many portions of the code is reused by the ransomware samples, bar the part which takes input (i.e., the password) from the user. Alternatively, the `Perseus` variant uses an embedded password to derive key, while the `CloudSword` variant uses the `System.Random` to generate a password from which the key is derived. The `CloudSword` sample even contains the exact error messages as in the full project at [180]; the `Perseus` sample uses the same code portions as in the tutorial, that without error messages.

Discussion. From our findings, we can conclude that certain ransomware engineers do copy-and-paste code from public sites. Surely, this conclusion cannot be representative of how all ransomware variants are coded. We do not even know whether who took advantage the public resources are professionals or amateurs, and it may be inherently hard to investigate for an answer on this matter due to the difficulty to reach out ransomware developers. However, we speculate, ransomware engineers are likely not in a different position than security developers. In [185], it is reported that in a population of three hundreds developers among which also professionals, only a quarter relied on the official documentation, while the rest consulted "the Internet", inevitably relaying in their code errors naïvities "out there", cause them to introduce security vulnerabilities in their code.

This seems to remain valid in our case: the security of some ransomware depends, at least in part, on the security reliability of the unofficial sources. A question remains open. Has the code-use helped ransomware criminals? The question is intertwined with the practice of dual-use of research in the field and, for this reason, we looked into the recent history of ransomware attacks in search for episodes of code re-use.

Dual-Use & Ransomware

Article 2 of Council Regulation (EC) No 428/2009 defines 'dual-use items' as items which can be used for both civil and military purposes. The article includes "Computers" and "Telecommunications and Information security" as categories to be screened for potential dual-use.

When it comes to cryptography, dual-use is a serious matter. In response to the US Munitions List, Category XIII, Materials and Miscellaneous Articles, which mentions "cryptographic devices, software and components", in a T-shirt shown at a DEFCON conference it was reported provocatively a piece of (encryption) code with the comment "this [code] can also be a munition" [187].

Within the cryptography community there is awareness that dual-use comes with a moral burden. Rogaway wrote that "cryptography is an inherently political tool, and it confers on the field an intrinsically moral dimension" [188]. Rogaway's argument is scoped in the contention between privacy on one side and mass surveillance on the other, but the message on that DEFCON T-shirt extends, even reverses, the matter. It raises the stake by pointing out that cryptographic code can be misused as a *weapon*. This is still the vision in certain countries, for instance the US, where non-military cryptography exports are if not forbidden at least controlled.

Listing 9.5: A simple function to encrypt files with a password, published at CodeProject. Contrary to the common practices, e.g., PBKDF2 [186], encryption key is derived directly using UTF-16 character encoding. In addition, instead of generating a unique value, encryption key is used as IV.

```
private void EncryptFile(string inputFile, string outputFile) {
    try {
        // Your Key Here
        string password = @"myKey123";
        UnicodeEncoding UE = new UnicodeEncoding();
        byte[] key = UE.GetBytes(password);

        string cryptFile = outputFile;
        FileStream fsCrypt = new FileStream(cryptFile, FileMode.
    Create);

        RijndaelManaged RMCrypto = new RijndaelManaged();
        CryptoStream cs = new CryptoStream(fsCrypt, RMCrypto.
    CreateEncryptor(key, key), CryptoStreamMode.Write);
        FileStream fsIn = new FileStream(inputFile, FileMode.Open)
    ;

        int data;
        while ((data = fsIn.ReadByte()) != -1)
            cs.WriteByte((byte)data);

        fsIn.Close();
        cs.Close();
        fsCrypt.Close();
    }
    catch {
        MessageBox.Show("Encryption failed!", "Error");
    }
}
```

Being the subject of this chapter ‘ransomware’, the matter must be contextualized: what about dual-use for cryptographic ransomware? And are ransomware and their cryptographic components weapons? To answer this question we look into cases of dual-use in ransomware. The most controversial is that of `HiddenTear` and its clones.

HiddenTear and its Clones. In 2015, a programmer Utku Şen published the first fully-fledged, open-source ransomware `HiddenTear`. This is the sample we commented in the previous section and whose code to generate a password is shown in Listing 9.3.

From the early days, the release of `HiddenTear` prototype received criticisms from the security community [189]. The main concern of the researchers is that even novice programmers can also make use of the published ransomware code while developing new variants. Time showed that they were right. A McAfee researcher stated that “in June (2017) almost 30% of the ‘new’ ransomware species we discovered was based on the `HiddenTear` code” [190].

Three months after the first release, Şen claimed that he wished (i) to provide an example of ransomware for beginners (ii) to build a honeypot for script kiddies [183]. It was partly true that the first variants of `HiddenTear` contained the same critical bugs that enabled the recovery of files [184]. However, one real thing in the malware history is evolution. The bugs in the original `HiddenTear` was fixed, and `HiddenTear` variant replaced the cryptographically insecure key generation method with a new one [191]

```
private static string GetEncKey() {
    try {
        using (WebClient webClient = new WebClient())
            return webClient.DownloadString(@"http://ohad.000
    webhostapp.com/cnc.php?txt=saveme").Trim();
    }
    catch {
        return "myke123!";
    }
}

private static void EncryptFile(string inputFile, string
    outputFile, string password) {
    try {
        byte[] bytes = new UnicodeEncoding().GetBytes(password);
        FileStream fileStream1 = new FileStream(outputFile,
    FileMode.Create);
        RijndaelManaged rijndaelManaged = new RijndaelManaged();

        // [...]

        fileStream1.Close();
        System.IO.File.Delete(inputFile);
    }
    catch {
        Console.WriteLine("Error: Encryption failed!");
    }
}
```

Listing 9.6: File encryption function of the Crypren sample. If C&C server is not reachable, the embedded password is used to derive keys. The resemblances between hard-coded passwords, key derivation methods and error messages are remarkable.

which evades the state-of-the-art key-oriented anti-ransomware defenses. Later, Şen admitted that his experiment was a total failure.

Another criticism to publishing the full source codes of a ransomware regards the principle of responsible disclosure. Prior to sharing the sources, Şen did not informed the anti-virus vendors. It should be noted that, when `HiddenTear` was released, on August 2015, only a few anti-ransomware systems existed: signature-based detection was the main technique to stop ransomware, just as the other malware types. Since `HiddenTear` and its variants were previously unseen, they were not recognized by AVs and therefore could run undetected for a while. The only precaution Şen took was putting a warning message in `HiddenTear` source code, which cyber-criminals could easily ignore.

Further Public Prototypes. Şen is not the only person that published a full ransomware prototype. There are several ransomware projects in different programming languages, publicly available on the Internet. For instance, `Arescrypt` is another open source ransomware implemented in C# [192]. `GonnaCry` is a Linux ransomware, implemented in both C and Python [193]. Aiming at web servers, a ransomware script written in PHP is also available at [194]. There is even an "academic" ransomware prototype implemented in Go language [195]. All these projects are publicly available at GitHub, a well-known platform among software developers. Moreover, although Şen abandoned the `HiddenTear` project, there are still several clones of the original repository and even some improved versions of `HiddenTear` on GitHub website, for example [196].

Zaitsev followed a different strategy when publishing `CryptoTrooper` [197]. He shared the core part of the prototype as a closed source binary. The encryption algorithm, whose code was not shared, contained a cryptographic flaw which enabled the recovery. Being closed source, the flaw in the encryption module of `CryptoTrooper` could not be fixed by the script-kiddies. Still, the community was divided: some found the idea useful, others did not [198]. In the end, Zaitsev removed the project from GitHub but, as in the case of `HiddenTear`, `CryptoTrooper` was forked by other developers. It is still accessible via various repositories.

All the developers of the publicly available ransomware prototypes states that their main motivation was *educational*. However, a well documented ransomware code would also help to-be-cyber-criminals to enter the ransomware business. Since ransomware prototypes remain available on the Internet, the ethical question here is whether security researchers need to publish and share full ransomware codes without feeling accountable of the consequences, a recognized ethical issue.

9.3 Ransomware Intelligence

Herr and Rosenzweig suggest that a piece of code is cyber-weapon when it combines "propagation, exploitation, and payload [i.e., damaging] capabilities" [187]. Each components, despite innocuous in separation, carry the potentiality to be combined with the missing others into a weapon. However, to have a military use, a software ' must create or tangibly support the deployment of destructive effects. These could be short term, where deleted data is restored from backup, or near permanent, where a payload is designed to damage a device's firmware" [187].

Ransomware may have such a destructive effect. For sake of an example, at the time of the writing, June 2019, the major electricity supplier in South Africa's city of Johannesburg was attacked, leaving more than a quarter of a million people in the dark. Another attack forced a shutdown of its websites and billing systems as a precautionary measure.

Ransomware variants, called *wipeware*, can wipe data clean. Allegedly deployed to attack Saudi energy companies and Iranian oil companies, they had destructive consequences. One variant of it, *Shamoon wiper*, has been released to attack Sony Pictures Entertainment, succeeding to avoid the outing of 'The Interview', a documentary mocking the North Korean dictator, Kim Jong-un. If we adhere to Schmid's claims that "terrorist violence is predominantly political" [199], such events can be considered also "terrorist attack" .

If ransomware are to be regarded as cyber-weapons, as we claim, could it be conceivable to apply intelligence and counter-intelligence strategies to mitigate the threats and control the consequences of an attack? And, if yes, how?

Cyber-Intelligence has been defined as "the process by which specific types of information important to national security are requested, collected, analyzed, and provided to policymakers, the products of that process"[200]. Duvenage *et al.* [201], call this *positive intelligence*, to distinguish it from *counter-intelligence*, which is the countering of an hostile intelligence activity.

Ransomware Positive Intelligence. For ransomware threat, positive intelligence could consist in gathering information about modalities of working. It should be about how the ransomware propagates, exploits vulnerabilities, and executes its payload. In the Open Source Intelligence (OSINT), several initiatives exist aiming to collect and analyze information gathered from public or open sources. An example is the *No More Ransom* project[4] (see also Chapter 1). It aims to inform the public and to collect incidents reports, including to gather the information from public platforms that can be potentially utilized by ransomware authors. Other platforms, although not specifically dedicated to ransomware, such as the *Malware Information Sharing Platform (MISP)*[5]—a free and open source software helping information sharing of threat intelligence, including cyber-security indicators—can offer tools that enable intelligence analysis. Such platforms can be employed to control the information flow during an attack, spreading alerts following a Warning and Coordination action, and to help potential victims "raise their shields" as soon as possible.

4: `https://www.nomoreransom.org`

5: `https://www.misp-project.org/`.

Ransomware Counter-Intelligence. According to [202], Counter Cyber Intelligence (CCI) is the ensemble of "all efforts made by one intelligence organization to prevent adversaries, enemy intelligence organizations or criminal organizations from gathering and collecting sensitive digital information or intelligence about them via computers, networks and associated equipment". It can be implemented using strategies that, according to Panda Security, a cyber-security company, either consists of "leaving doors open" (i.e., left access points unprotected on purpose), "inject fake information" (i.e., fake confidential information), and "keeping them busy while stealing" (i.e., watching and obtaining information about the attacker).

Looking into the internet and searching for "counter-intelligence for ransomware", we have found that the majority of the initiatives to protect from ransomware attacks focuses on raising awareness. For instance, the US National Counter-intelligence and Security Center (NCSC) has launched in January 2019 a campaign "Know the Risk, Raise Your Shield". The Cybersecurity and Infrastructure Security Agency (CISA) addresses ransomware specifically, but it is all about knowing the threat and apply general security best practices such as backing-up data. We have found, within the scope of cryptographic ransomware and limited to this on-going work, nothing about "leaving the door open", "inject fake information", or ‘’keep them busy".

The second measure (i.e., "inject fake information") may not be fully applicable at least if that means to avoid to spread knowledge about how to build the ransomware weapon: the instruments of cryptography are nowadays already known and public. However, at the light of what we have discussed in the previous sections, it may be a strategy to post the code of variants whose decryptors already exist.

For what concerns the "keep them busy" paradigm, as discussed in [40], it may be possible the use decoy files to deflect a ransomware attack against irrelevant (for the victim) files, so gaining that amount of time required to stop the attack's development. Using decoy files could be paired with strategies that downgrade the efficiency of encryption for applications that are not trustworthy or whitelisted. We have not investigated in this direction, but this option seems preferable to that of running untrusted application in sandbox. This can be less effective, since certain ransomware sample recognize the presence of a virtual environment and remain dormant. A few articles suggest the use of Artificial Intelligence (e.g., [203]), but we did not look into this direction.

Conclusion

Closing Remarks and Future Work 10

In the last few years cryptographic ransomware attacks have dominated the cyberthreats landscape [204]. They target the most valuable asset of today's computer users and companies — the data. Cryptographic ransomware encrypts the data which become inaccessible to their owners. If, during this task, the ransomware uses strong cryptography, it is unfeasible for the legitimate file-owner to recover the files' contents without the decryption key. Victims, users and companies alike, are then forced to pay a ransom if they want to regain access to their data[1].

1: Paying the ransom does not ensure getting the files back. Some statistics show that 67% of the companies who paid the ransom could recover their data. In 2018, it was even worse, only the half could. For more details, see [2].

The quick return in revenue together with the practical difficulties in the accurate tracking of cryptocurrencies, used to perform the ransomware payment by victims, have made ransomware a preferred tool for cybercriminals. In particular, exploiting zero-day vulnerabilities found in Windows OS, the most-widely used OS on desktop computers[2], has enabled ransomware to extend its threat and damage at world-wide level. For instance, `WannaCry` and `NotPetya` have affected almost all countries, impacted organizations, and the latter alone caused damage which costs more than \$10 billion [8].

2: According to StatCounter, Windows usage is 77.21% as of August 2020 [205].

Security researchers have worked to slow down the threat. While these systems —without any doubt— increased the bar for cybercriminals, most of them are designed by adopting the existing behavioral analysis methods for ransomware, rather than approaching the problem from a cryptographic perspective. Attackers use the power of cryptography, we believe the defenders must, too. Therefore, we defined the goal of our research to fill this gap:

Research Goal: *To study the behavior of ransomware, understand its weaknesses, and uncover the cryptographic roots of it to design a defense system which advances the state of the art.*

Our research contributes to the defense against cryptographic ransomware. It sheds light on the state-of-the-art of both sides in the ransomware ecosystem —attackers and defenders. Investigating the phenomenon from these two point of views provides a comprehensive understanding of the ransomware, and unveils its strengths and weaknesses. The result of this thesis establishes an enhanced protection from cryptographic ransomware. It does so by: (1) analyzing the inner mechanisms, operation flow and other cryptographic aspects of a ransomware attack, in particular, key management; (2) unveiling the issues and challenges in current defense proposals, i.e., the flaws and vulnerabilities therein; (3) designing a defense system that is based on the principles of the modern cryptography. In the following, we review our research questions and explain how we answered them.

How can future generations of ransomware may work to bypass current anti-ransomware systems? The purpose of this part is to warn the scientific community of forthcoming ransomware threats. By talking about how six cutting-edge anti-ransomware solutions —at the time of this thesis research, implementing strategies of key escrow, and behavioral analysis are the most advanced strategies known against active ransomware samples— could be overthrown by smarter and more sophisticated malware, we hoped to have revealed what strategies those malware could trying to implement, so indicating where anti-ransomware engineers have to focus their efforts. Since it is believed that the ransomware threat will increase not in number of attacks but in sophistication, to keep anti-ransomware ideas ahead of time may be a game-changing factor.

That said, malware mitigation is an arms race and we expect new generations of ransomware coming soon with renovated energy and virulence, adapting their attack strategies to challenge current defenses. New variants of ransomware have been observed constantly during the last years. Those called *scareware* prefer to exploit people's psychology, threatening them into pay the ransom without, however, doing any serious encryption: despite deceitful they are technically benign applications. Others, however, will be variants of real cryptographic ransomware and able to overcome control and to encrypt a victim's files using strong encryption. A white paper by Symantec [3] reports that ransomware is becoming instrument for specialists and targeted attack groups, a weapon not only to extort money but to cover up other attacks and, when using strong encryption, used in fact as a disk wiper. It is to this latter category that our research is dedicated. As security professionals we feel compelled to be prepared to face forthcoming threats thus to identify and anticipate potentially dangerous ransomware variants, and warn the scientific community about them.

Another defense approach, decoy-based strategies have been successfully used in providing evidence of an intrusion into a computer system. They have been called in different ways, the most common ones using the prefix 'honey-' as in honey pot, honey words, honey files, and honey token. Their use against malware, such as ransomware, is however still in its infancy, and there is little evidence that mitigating strategies that have worked against human intruders might work against ransomware. From one side, some applications may lack certain specific features that are usually exploitable to lure a human adversary into committing false steps – this makes malware immune to certain decision bias and vulnerabilities; from the other side, as the ransomware is running in the host system it might have access to additional capabilities, e.g. that of spying file activities, that are not available to a system intruder.

In this thesis we have looked into what limits decoy strategies may encounter when applied against ransomware. We first address the issue from a theoretical point of view, and then we have described a practical proof-of-concept that shows how some existing decoy-based solutions can be easily defeated. The results of our experiments show that we need to re-design the generation of honey documents so that their use against future ransomware will be as effective as their use against human intruders is. Our findings also provide the opportunity of investigation for two future directions. In the first one, an hypothetical strong adversary may be recognized and stopped by using other complementary strategies than those based on decoy files, and the research question is how to effectively combine different anti-ransomware strategies. In the second one, any anti-ransomware that relies on decoy files has to consider its usability, a quality that we have proposed be measured in terms of confoundedness, that is, how probably is that decoy files confuse a honest user into accessing them.

We are aware that the research we have ourselves embarked may give ideas to criminals. But there is no reason to believe that criminals will not have those ideas by themselves. In the history of malware (see e.g., [53]) criminals have always tried to be one step ahead; besides, our research has nothing fancy and it does not contain such an inventive step that cannot be reproduced by others. It more humbly roots into how cryptography works. However, even with this premise, we questioned ourselves about how to answer this research question and the following one ethically, in Chapter 2.

How secure are the real world protection mechanisms against novel ransomware attacks? AV programs have become one of the *de facto* computer security standards. Several companies trust AVs to protect their assets without questioning how AVs do their job. In their turn, AVs have their assumptions, we presume, and trust their security mechanisms be solid. Sometimes, we learned, also that trust is not questioned further. This is probably necessary in the fast-paced cat-and-mouse game in which AVs and malware are engaged, always running one after the other; but we, as researchers, can question the robustness of certain assumptions. In particular, we questioned whether built-in white-listed applications can be undetectably be manipulated and instructed as well as undetectably to do harm to user files. For instance, we tried to see whether they can be used to encrypt a file's content or to wipe it out. We also questioned whether AV's real-time scanning protection feature can be turned-off by a malware that simulates mouse and keyboard events without being caught while doing so. Surprised ourselves, we succeeded in proving that these vulnerabilities exist for quite a number of AVs. What we found is indeed surprising in general, considering that almost all AVs have today ransomware detection modules.

The security issues we discovered reveal vulnerabilities both in the extension in which certain security mechanisms are supposed to operate, and in the they way in which the interaction between the OS and the AV defenses is believe to work. The vulnerabilities we discuss in this thesis are therefore not implementation flaws. To give substance to our findings, we have designed and implemented two proof-of-concept programs, *Ghost Control* and *Cut-and-Mouse*, which are able to fully disable the run-time protection of several consumer AVs, and/or to bypass their defenses in protection of user files against threats like ransomware. We tested them against the current most comprehensive list of consumer AV products. Not all of them are vulnerable but for those which are we also speculated about possible fixes. These require software developers to have a general understanding of what caused them. We stated that understanding in a potentially new, or at least renewed, *security principle.*

One could question whether such *Cut-and-Mouse* and *Ghost Control* attacks can, after all, be detected by the human user's who sees e.g., the mouse pointer moving and clicking here and there. Perhaps, users can be puzzled. Still they are like not be able to react promptly to stop the attack: users are notoriously bad on implementing security measures because security is not their primary goal. Thus making security dependent on the user's reaction to something strange on his screen is fundamentally not a solution and it does not give more security guarantees.

We have also found that *Cut-and-Mouse*'s and *Ghost Control*'s working principle, that of using mouse and keyboard events, works even when AVs run them in a sandbox, revealing that the sheer use of a sandbox is not suffice to protect a system from certain malware: mouse clicks are not filtered and therefore can escape the sandbox.

In addition, malware can perform these attacks when the user is not using the computer, e.g., through some heuristic based on user's activities. Thus a better mitigation solution would be aimed at understanding whether keyboard and mouse events come from a legitimate user or whether instead they are synthesized by a (malicious) program. In a sense, discerning such situation is what malware is already trying to achieve, namely understanding if it is running in a sandbox, e.g., using reverse Turing tests to detect the presence (or absence) of a human, – this further reinforces the analogy of attackers and defenders are each learning from others.

Before that discernment becomes possible, OS and AV defences have to cooperate better. At the root of our findings there is a misalignment between two different concepts: that of integrity levels used by the OS, and that of trusted applications on which instead AV defences rely upon. They have not been conceived to work together and, at a higher level, they have to be harmonized. This is indeed what our Principle 1 means to achieve.

The arms-race between malware authors and anti-malware developers has been occurring in the digital world since its early days. In the last two decades, however, malware developers are living their golden age. Powerful techniques such as metamorphic obfuscation have largely limited the effectiveness of signature-based defense systems. Consequently, protection systems started to move toward dynamic analysis to detect freshly generated malware. Sandboxes, isolated execution and monitoring environments, have been widely used for this purpose. Consequently, the sandbox environments have been among the top items of the target list of cybercriminals, and numerous advanced evasion techniques have been seen in real-world malware attacks.

Finally, we described a not-yet-discussed evasion technique that we found being followed by a still active `TeslaCrypt` ransomware sample. Using it, the sample manages to look benign and therefore is capable to avoid being classified as a malware, when analyzed in Cuckoo sandbox. We suggested two improvements, one tightening current behavioral analysis methods, the other working to mitigate the severity of an attack by that sample. Namely, we discussed (i) a smart execution strategy to increase the detection chance of the malware in the sandbox; and (ii) a complementary defense method to be employed in the actual user environment, preferably integrated into already existing protection systems. We have also compared pros and cons of the two defense approaches and their potential side effects to the user.

Can ransomware be stopped by controlling secure randomness sources? Cryptographic ransomware is a modern global crime and a large amount of public and private institutions have been attacked already. The problem is that encryption is a powerful tool in the hands of criminals, hard to fight. By encrypting critical files on the victim's machine, ransomware blocks access to information and compromises critical services, wreaking an economical and social havoc because, unless victims pay the demanded ransom to receive the correct decryption key, they might not be able to recover their files if no backup is available. Computational complexity results ensure that a properly implemented encryption is irreversible, but to realize this theoretical result in practice, ransomware has to use cryptographically secure encryption keys. Many variants choose weaker alternatives: although there could be a theoretical solution to reverse their encryption at affordable costs, such *scareware* succeed in persuading victims to pay. Other variants, implement a theoretically weak but good-enough encryption to make decryption-without-the-key sufficiently painful to convince that paying the ransom is the lesser of two evils.

But in the restricted niche of ransomware that want their damage to be computationally irreversible, one finds the most disruptive variants, for instance, `WannaCry`, `Petya`, `GoldenEye`, `CryptoLocker`, `Locky`, `Ryuk`, `Samsam`, and `NotPetya`. These ransomware families need a good source of random numbers and all of them find it in the CSPRNG available on a victim's system. Today, such functions are indeed reliable and *de facto* source of cryptographic randomness available on a computer.

To contain the threat coming from ransomware in this cryptographically strong niche, we proposed to control access to CSPRNG APIs. We proved the concept by stopping a very large class of real active ransomware from doing any damage to any file —remarkably including `NotPetya`, which was till that moment believed unstoppable. But a concept, as much as promising can be, is not yet a fully-fledged application. Discussing how to implement it into an effective anti-ransomware defense, called NoCry, is what we have done next. We solved several critical security and design issues: how to ensure that the attack surface of the architecture is reduced; how to bootstrap the Whitelist DB, honest cryptographic applications calling CSPRNG APIs and maintain it with a minimal user intervention, arguably resulting in a very low false positive rate; how to reduce the overhead that the access control imposes on the systems performance to a negligible amount. By not relying on any IPC, we removed any known-to-be-vulnerable elements from the architecture, so addressing the first issue; we addressed the last, by a better decision making that drastically improves the overhead. With respect to the previous proof-of-concept, reducing it from several thousands percent down to about 20%: quantified, the overhead is now a few *hundredth of a second.*

We are aware that there is no silver bullet for ransomware mitigation. Each defense system has pros and cons, and NoCry may well find its beater in some next generation ransomware. As discussed in Chapter 3, ransomware applications can find other ways than calling CSPRNG to get random numbers e.g., by relying on non-cryptographic sources of randomness, but we believe that the alternative choices have weak points. The fact is that all the samples and variants of ransomware in the cryptographically-hard niche that we have analyzed so far, do call CSPRNG APIs. Thus, today, these functions are the most reliable source of randomness for application in search to build cryptographically strong encryption keys. And if in the future other functions will available for the same task, the fundamental question that remains to be solved is how many of these functions are, and whether by controlling access to these APIs, we can still implement a targeted strategy as the one in NoCry that enables a decision making with an arguably low false positive rate.

The approach described to answer the research question has been shown to be highly effective against the current generation of ransomware, but doubtless, (having read this thesis), the authors of ransomware will devise new strategies to evade our approach. The race between ransomware and anti-ransomware will continue.

What are the implications of emerging technologies on the ransomware phenomenon? Ransomware is a class of malware whose goal is to extort money, a goal that is facilitated by current anonymous currencies which enables cybercriminals to be paid without being traced. Then we need solid defense systems against what can easily degenerate in a pandemic of digital crimes. However, unlike conventional anti-malware systems, ransomware mitigation does not tolerate mistake. If the ransomware is implemented properly and the attack succeeds, then the damage taken may be irreversible.

Existing ransomware mitigation systems are build upon the analysis of collected samples but a better strategy is to anticipate the future, and be prepared for the ransomware that will come. In this respect, we described possible threats that ransomware may pose by relying on novel strategies, like acquiring rootkit capabilities, utilizing obfuscation and whitebox cryptography, not yet adopted in real attacks as well as by targeting critical domains, such as the Internet of Things and the Socio-Technical systems, which will worrisomely amplify the effectiveness of ransomware attacks. Our research is timely, since it is known that we must design products keeping security in mind, not integrating after whereas network infrastructures must be carefully configured and fully patched in order to prevent ransomware attacks through data exfiltration. We hope that our observations help developing and building more robust defense systems against ransomware threat.

Ransomware are emerging as cyber-weapons. They have been used in attacks that resemble actions of cyber-war, and are far more dangerous and disruptive than traditional malware. Consequently, the research community should reflect on coordinated actions to address the threat under an appropriate code of ethical conduct.

Having discovered that a few ransomware contain a copy-paste from cryptographic code available in public sources, we debated the matter of dual-use in cryptographic research and recalled (in)famous antecedents in the recent ransomware history. Since we managed to build decryptors for those ransomware, the dual use turned out to be a double-edge for the criminals, but generally it is not. After having build a case for ransomware as cyber-weapon, we briefly reviewed intelligence and counter-intelligent strategies that could be used in the fight against ransomware.

We did not backed our speculations with field studies or interviews. Ours is an educated argumentation, but its purpose is to invite the anti-ransomware community to be more proactive in the cyber-war against ransomware. Even the excellent *NoMoreRansom* project, which offers decryptors when they are available (as did in June 2019, with the then-latest version of `GandCrab`[3]), at the end of the day praises for keeping back-up, within a "Better Safe Than Sorry" advice.

3: Decryptors are available for `GandCrab` versions 1, 4, 5.01, and 5.2. For more details, see `https://www.malwarebytes.com/gandcrab/`.

10.1 Future Works and Open Problems

Throughout the development of this thesis research, we identified some future works and open problems that remain to be explored. We classified them in four main categories, as follows:

New Attack Techniques Modern ransomware uses well-known cryptographic algorithms, e.g., AES, ChaCha, for encrypting files. Data encrypted with these algorithms contain high entropy in average, causing a stark difference with that of the original file. Taking advantage of this disparity, current behavioral analysis systems use statistical tests to detect if a file is being encrypted by comparing the buffers before and after a file write request. We could evade this protection by using a pure permutation of byte arrays to obfuscate files, and this is definitely not as secure as those of standard ciphers. The weaknesses of pure permutation has been known by the cryptographic community but, to the best of our knowledge, has never been explored from ransomware evasion perspective.

Ransomware authors might have to remain limited to permutation, even if it is a quite weak way of encryption, to have at least a chance of extortion rather than being caught immediately by behavioral analysis systems which would identify the robust algorithms like AES. On the other hand, if the permutation can be discovered practically, the ransomware cannot force the victims to pay. However, the question is still open: does it provide the minimal security level in the context of ransomware, i.e., decryption might be possible but paying the ransom is more economic than decrypting the files? Continuing on this line of thought, one might ask the question if there would be a way to encrypt files securely while keeping the entropy of the file at the same level? Ransomware equipped with that technology might be able to evade existing behavioral analysis based anti-ransomware systems. Developing such an encryption algorithm, however, is far from a trivial task.

Key Generation Modern ransomware employs hybrid cryptosystems for scalability and efficiency reasons. Consequently, managing the encryption keys, especially the symmetric keys used to encrypt bulk data, in a secure manner is critical for a successful ransomware campaign, as a flaw in the transport, usage or storage of the keys might allow security professionals to build a decryptor. In particular, if the victims can obtain the keys used to encrypt files, decrypting the files without paying a ransom would be feasible. This is obviously against the goals of ransomware authors so they try to obtain encryption keys securely.

The security analyses of key generation methods of ransomware, both in this thesis and other works, show that the only option for current ransomware to get good encryption keys is to use a CSPRNG. However, thanks to UShallNotPass and its successor NoCry, ransomware can be prevented from accessing this critical resource effectively and efficiently.

In response to our defense systems, cybercriminals may develop new methods to generate strong encryption keys. To anticipate this evolution, we are also studying alternative ways to derive encryption keys, for instance from files as it is done in convergent encryption, a technique applied in cloud computing to build keys for symmetric algorithms. That said, the security of generating asymmetric keys from files is not studied yet, and we think it is an interesting research topic to investigate further. Furthermore, preventing other ways that ransomware could (i) use to generate encryption keys; and (ii) circumvent calls to CSPRNG in order to evade our controls is a research direction that must be explored.

Other future work still needs to be done. To build a practical and automatic whitelisting strategy with low false positive rates would increase the usability of NoCry, and we are investigating the potential ways for achieving for this goal. The argument that we have a reduced false positive rate in NoCry also has to be supported by experimental evidence. This means to run stress tests while running a generous number of various benign cryptographic applications under different conditions. Beyond having measured the overhead in terms of loss of performance, we still need to assess the user experience (UX) of NoCry running on different kinds of computers, included on battery powered mobile devices, to verify whether the overhead is imperceptible, as we claim, by users in their daily activities.

Better Decoys Deception-based anti-ransomware systems create decoy files among the genuine files of the user. This practice works under the assumption that current ransomware is decoy-blind so that in a ransomware attack, decoy files would be accessed which would lead to detection. Today this approach might seem secure, however, the actual algorithm to prepare decoy files is of uttermost importance as ransomware might develop decoy-aware features. Furthermore, algorithms to place decoy files must ensure that the number of files encrypted by a ransomware before a decoy gets accessed is minimum, of if possible, zero. Design and analysis of such algorithms must be studied to guarantee the security of the decoy-based defense systems.

Another gap in the decoy-files domain is the lack of research in their usability. Even though there exists several anti-ransomware systems using decoy files, both in academic literature and commercial market, none of them provides a usability report nor a field test. In other words, there is no scientific evidence about how much interruption decoy files cause. Finding the right balance for a decoy between being effective without confusing the user (who might then decide to switch off the defense, or change it for another) is a research challenge by itself that has to be addressed.

Secure Inter-Operation The security industry has a large number of vendors so that the consumers —including home users, businesses, governments— have a wide variety of options. That said, there is not a silver bullet to satisfy all kinds of protection against cyberthreats. Therefore, users might need to install and run security products of different vendors simultaneously on the same system. While each of these defense software might work against the targeted threat with a high success ratio, when considered individually, they might not cover the entire threat landscape. This is not a fault of the vendors; no tool alone can provide every security feature. However, tools should be capable of being orchestrated by the OS to fill all the gaps in the security surface. Otherwise, by passing through the non-controlled zones, malware could reach its target. In this thesis, we have demonstrated an evidence of this risk, and show how devastating the results could be in the absence of the cooperation between components of a system. Design, implementation, and evaluation of such a secure inter-operation infrastructure in the modern OS are important research tasks that have be performed.

Bibliography

[1] Symantec. *Internet Security Threat Report. Ransomware 2017.* White Paper. 2017. URL: https://docs.broadcom.com/doc/istr-ransomware-2017-en (visited on 08/18/2020) (cited on page 3).

[2] CyberEdge. *Cyberthreat Defense Report.* White Paper. 2020. URL: https://cyber-edge.com/wp-content/uploads/2020/03/CyberEdge-2020-CDR-Report-v1.0.pdf (visited on 08/18/2020) (cited on pages 3, 125).

[3] Symantec. *Internet Security Threat Report.* White Paper. 2018. URL: https://docs.broadcom.com/doc/istr-23-2018-en (visited on 08/18/2020) (cited on pages 3, 31, 126).

[4] Symantec. *Internet Security Threat Report.* White Paper. 2019. URL: https://docs.broadcom.com/doc/istr-24-2019-en (visited on 08/18/2020) (cited on page 3).

[5] Mark Ward. 'Cryptolocker victims to get files back for free'. In: *BBC News* (Aug. 6, 2014). URL: https://www.bbc.co.uk/news/technology-28661463 (visited on 08/18/2020) (cited on page 3).

[6] F-Secure. *The State of Cyber Security.* White Paper. 2017. URL: https://blog-assets.f-secure.com/wp-content/uploads/2019/10/18165954/Cyber_Security_Report_2017.pdf (visited on 08/18/2020) (cited on page 3).

[7] US Department of Justice. *How to Protect your Networks from Ransomware.* June 2016. URL: https://www.justice.gov/criminal-ccips/file/872771/download (cited on page 3).

[8] Andy Greenberg. *The Untold Story of NotPetya, the Most Devastating Cyberattack in History.* https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/, Last accessed on February 22, 2019. Aug. 2018 (cited on pages 3, 90, 91, 125).

[9] Jack Stubbs. *'Payment sent' - travel giant CWT pays $4.5 million ransom to cyber criminals.* July 31, 2020. URL: https://www.reuters.com/article/us-cyber-cwt-ransom/payment-sent-travel-giant-cwt-pays-4-5-million-ransom-to-cyber-criminals-idUSKCN24W25W (visited on 08/18/2020) (cited on page 3).

[10] David E. Sanger Manny Fernandez and Marina Trahan Martinez. 'Ransomware Attacks Are Testing Resolve of Cities Across America'. In: *The New York Times* (Aug. 22, 2019). URL: https://www.nytimes.com/2019/08/22/us/ransomware-attacks-hacking.html (visited on 08/18/2020) (cited on page 3).

[11] Steve Morgan. *Global Ransomware Damage Costs Predicted To Reach $20 Billion (USD) By 2021.* Oct. 21, 2019. URL: https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-20-billion-usd-by-2021/ (visited on 08/18/2020) (cited on page 3).

[12] McAfee. *McAfee Labs Threats Report.* White Paper. Aug. 2019. URL: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf (visited on 08/18/2020) (cited on page 4).

[13] 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 (cited on pages 4, 103).

[14] U.S. Federal Bureau of Investigation. *High-Impact Ransomware Attacks Threaten U.S. Businesses And Organizations.* Oct. 2, 2019. URL: https://www.ic3.gov/media/2019/191002.aspx (visited on 08/18/2020) (cited on page 4).

[15] Brianna Gammons. *4 Surprising Backup Failure Statistics that Justify Additional Protection.* Jan. 2017. URL: https://web.archive.org/web/20180825141202/https://blog.barkly.com/backup-failure-statistics (visited on 08/18/2020) (cited on page 4).

[16] Alexandre Gazet. 'Comparative analysis of various ransomware virii'. In: *Journal in Computer Virology* 6.1 (Feb. 2010), pp. 77–90 (cited on page 5).

[17] Brian Baskin. *TAU Threat Discovery: Conti Ransomware.* July 8, 2020. URL: https://www.carbonblack.com/blog/tau-threat-discovery-conti-ransomware/ (visited on 08/18/2020) (cited on page 5).

[18] Sara Tilly. *Cryptolocker Prevention – How to secure your server environment.* Mar. 2017. URL: https://blog.syskit.com/cryptolocker-prevention (cited on page 5).

[19] Carl Woodward and Raj Samani. *Is WannaCry Really Ransomware?* June 2017. URL: https://securingtomorrow.mcafee.com/executive-perspectives/wannacry-really-ransomware/ (cited on page 6).

[20] Michael Young and Ryan Zisk. *Decrypting the NegozI Ransomware.* Sept. 2017. URL: https://yrz.io/decrypting-the-negozi-ransomware (cited on page 6).

[21] MalwrPost. *Technical Analysis of Rush/Sanction Ransomware.* Apr. 2016. URL: https://malwrpost.wordpress.com/2016/04/06/technical-analysis-of-rush-sanction-ransomware/ (cited on page 6).

[22] Kevin Savage, Peter Coogan, and Hon Lau. *The evolution of ransomware.* Tech. rep. Symantec, Aug. 2015. URL: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-evolution-of-ransomware.pdf (cited on page 6).

[23] N. Scaife et al. 'CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data'. In: *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS).* June 2016, pp. 303–312 (cited on pages 6, 22, 23, 30, 84, 86, 91, 109).

[24] Vassil Roussev. 'Data Fingerprinting with Similarity Digests'. In: *Advances in Digital Forensics VI.* Ed. by Kam-Pui Chow and Sujeet Shenoi. Berlin: Springer, 2010, pp. 207–226 (cited on pages 6, 22).

[25] Andrea Continella et al. 'ShieldFS: A Self-healing, Ransomware-aware Filesystem'. In: *Proceedings of the 32nd Annual Conference on Computer Security Applications.* ACSAC '16. New York, NY, USA: ACM, 2016, pp. 336–347 (cited on pages 6, 22, 91, 109).

[26] Amin Kharraz and Engin Kirda. 'Redemption: Real-Time Protection Against Ransomware at End-Hosts'. In: *Research in Attacks, Intrusions, and Defenses.* Ed. by Marc Dacier et al. 2017, pp. 98–119 (cited on pages 6, 22, 86, 91, 109).

[27] Aurélien Palisse et al. 'Data Aware Defense (DaD): Towards a Generic and Practical Ransomware Countermeasure'. In: *Secure IT Systems.* Cham: Springer, 2017, pp. 192–208 (cited on pages 7, 22, 23).

[28] Aurélien Palisse et al. 'Ransomware and the Legacy Crypto API'. In: *The 11th International Conference on Risks and Security of Internet and Systems - CRiSIS 2016.* Ed. by Frédéric Cuppens et al. Risks and Security of Internet and Systems. Springer, Sept. 2016, pp. 11–28 (cited on pages 7, 21, 23).

[29] Kyungroul Lee, Insu Oh, and Kangbin Yim. 'Ransomware-Prevention Technique Using Key Backup'. In: *Big Data Technologies and Applications.* Springer International Publishing, 2017, pp. 105–114 (cited on pages 7, 21, 23).

[30] Eugene Kolodenker et al. 'PayBreak: Defense Against Cryptographic Ransomware'. In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security.* ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 599–611 (cited on pages 7, 21, 23, 91, 100, 109, 111).

[31] H. Kim et al. ‘Dynamic ransomware protection using deterministic random bit generator’. In: *2017 IEEE Conference on Application, Information and Network Security (AINS)*. Nov. 2017, pp. 64–68 (cited on pages 7, 21, 23).

[32] Jeonghwan Lee, Jinwoo Lee, and Jiman Hong. ‘How to Make Efficient Decoy Files for Ransomware Detection?’ In: *Proc. of Int. Conf. on Research in Adaptive and Convergent Systems*. RACS ’17. Krakow, Poland: ACM, 2017, pp. 208–212 (cited on pages 7, 34, 38, 43).

[33] Shagufta Mehnaz, Anand Mudgerikar, and Elisa Bertino. ‘RWGuard: A Real-Time Detection System Against Cryptographic Ransomware’. In: *Research in Attacks, Intrusions, and Defenses*. Cham: Springer International Publishing, 2018, pp. 114–136 (cited on pages 7, 22, 34, 37, 43).

[34] WatchPoint Data. *CryptoStopper*. 2018. URL: `https://www.watchpointdata.com/cryptostopper` (cited on pages 7, 34, 37).

[35] Felix Gröbert, Carsten Willems, and Thorsten Holz. ‘Automated Identification of Cryptographic Primitives in Binary Programs’. In: *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*. RAID’11. Menlo Park, CA: Springer-Verlag, 2011, pp. 41–60 (cited on page 7).

[36] Pierre Lestringant, Frédéric Guihéry, and Pierre-Alain Fouque. ‘Automated Identification of Cryptographic Primitives in Binary Code with Data Flow Graph Isomorphism’. In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS ’15. ACM, 2015, pp. 203–214 (cited on page 8).

[37] Xin Luo and Qinyu Liao. ‘Awareness Education as the Key to Ransomware Prevention’. In: *Information Systems Security* 16.4 (2007), pp. 195–202 (cited on page 8).

[38] Raj Samani and Christiaan Beek. *An Analysis of the WannaCry Ransomware Outbreak*. May 12, 2017. URL: `https://www.mcafee.com/blogs/enterprise/analysis-wannacry-ransomware-outbreak/` (visited on 08/18/2020) (cited on page 8).

[39] Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. ‘Next Generation Cryptographic Ransomware’. In: *Secure IT Systems - 23rd Nordic Conference, NordSec 2018, Oslo, Norway, November 28-30, 2018, Proceedings*. Vol. 11252. LNCS. Springer, Nov. 2018, pp. 385–401 (cited on page 11).

[40] Ziya Alper Genç, Gabriele Lenzini, and Daniele Sgandurra. ‘On Deception-Based Protection Against Cryptographic Ransomware’. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2019, pp. 219–239 (cited on pages 11, 121).

[41] Ziya Alper Genç, Gabriele Lenzini, and Daniele Sgandurra. ‘A Game of “Cut and Mouse”: Bypassing Antivirus by Simulating User Inputs’. In: *Proceedings of the 35th Annual Computer Security Applications Conference*. ACSAC ’19. San Juan, Puerto Rico: Association for Computing Machinery, 2019, pp. 456–465 (cited on page 11).

[42] Ziya Alper Genç, Gabriele Lenzini, and Daniele Sgandurra. ‘Cut-and-Mouse and Ghost Control: Exploiting Antivirus Software with Synthesized Inputs’. In: *Digital Threats: Research and Practice* (). In press (cited on page 11).

[43] Ziya Alper Genç, Gabriele Lenzini, and Daniele Sgandurra. ‘Case Study: Analysis and Mitigation of a Novel Sandbox-Evasion Technique’. In: *Proceedings of the Third Central European Cybersecurity Conference*. CECC 2019. Munich, Germany: Association for Computing Machinery, 2019 (cited on page 12).

[44] Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. ‘No Random, No Ransom: A Key to Stop Cryptographic Ransomware’. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Cham: Springer International Publishing, 2018, pp. 234–255 (cited on pages 12, 85).

[45] Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. ‘Security Analysis of Key Acquiring Strategies Used by Cryptographic Ransomware’. In: *Proceedings of the Central European Cybersecurity Conference 2018*. CECC 2018. Ljubljana, Slovenia: ACM, 2018, 7:1–7:6 (cited on pages 12, 103).

[46] Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. ‘NoCry: No More Secure Encryption Keys for Cryptographic Ransomware’. In: *Emerging Technologies for Authorization and Authentication*. Cham: Springer International Publishing, 2020, pp. 69–85 (cited on page 12).

[47] Ziya Alper Genç, Gabriele Lenzini, and Peter Y. A. Ryan. ‘The Cipher, the Random and the Ransom: A Survey on Current and Future Ransomware’. In: *Proceedings of the Central European Cybersecurity Conference 2017*. CECC 2017. Ljubljana, Slovenia: University of Maribor Press, 2017, pp. 89–102 (cited on page 12).

[48] Ziya Alper Genç. and Gabriele Lenzini. ‘Dual-use Research in Ransomware Attacks: A Discussion on Ransomware Defence Intelligence’. In: *Proceedings of the 6th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,* INSTICC. SciTePress, 2020, pp. 585–592 (cited on page 12).

[49] Council of European Union. *Council regulation (EU) no 428/2009*. https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex:32009R0428, Last accessed on February 22, 2019. 2009 (cited on page 15).

[50] European Commission. *Guidance note – Research involving dual-use items*. http://ec.europa.eu/research/participants/data/ref/h2020/other/hi/guide_research-dual-use_en.pdf, Last accessed on February 22, 2019 (cited on page 15).

[51] Working Group Dual Use of the Flemish Interuniversity Council. *Guidelines for researchers on dual use and misuse of research*. 2017 (cited on page 15).

[52] Alana Maurushat. *Disclosure of Security Vulnerabilities: Legal and Ethical Issues*. London: Springer-Verlag London, 2013 (cited on page 15).

[53] Jessica R. Herrera-Flanigan and Sumit Ghosh. ‘Criminal Regulations’. In: *Cybercrimes: A Multidisciplinary Analysis*. Ed. by Sumit Ghosh and Elliot Turrini. Berlin, Heidelberg: Springer, 2011. Chap. 16, pp. 265–308 (cited on pages 17, 127).

[54] J. P. Sullins. ‘A Case Study in Malware Research Ethics Education: When Teaching Bad is Good’. In: *2014 IEEE Security and Privacy Workshops*. May 2014, pp. 1–4 (cited on page 17).

[55] Ronald Deibert and Masashi Crete-Nishihata. ‘Blurred Boundaries: Probing the Ethics of Cyberspace Research’. In: *Review of Policy Research* 28.5 (2011), pp. 531–537 (cited on page 17).

[56] Phillip Rogaway. *The Moral Character of Cryptographic Work*. Cryptology ePrint Archive, Report 2015/1162. https://eprint.iacr.org/2015/1162. 2015 (cited on page 17).

[57] Directorate-General for Research and Innovation. *Ethics for researchers*. Tech. rep. KI-32-13-114-EN-N. European Commission, July 2015 (cited on page 18).

[58] Ian Darwin. *Fine Free File Command*. http://www.darwinsys.com/file/ (cited on page 22).

[59] Fabio De Gaspari et al. *The Naked Sun: Malicious Cooperation Between Benign-Looking Processes*. 2019. arXiv: 1911.02423 [cs.CR] (cited on page 22).

[60] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Berlin, Heidelberg: Springer-Verlag, 2002 (cited on page 23).

[61] Stijn Pletinckx, Cyril Trap, and Christian Doerr. ‘Malware Coordination using the Blockchain: An Analysis of the Cerber Ransomware’. In: *IEEE Conference on Communications and Network Security, CNS 2018*. Piscataway, New Jersey, US: IEEE, 2018, pp. 1–9 (cited on pages 24, 77).

[62] John R. Douceur et al. ‘Reclaiming Space from Duplicate Files in a Serverless Distributed File System’. In: *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS’02)*. ICDCS ’02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 617–624 (cited on pages 25, 93).

[63] Benyamin Hirschberg et al. *Ransomware Key Extractor and Recovery System*. `https://patentscope.wipo.int/search/en/detail.jsf?docId=US215058675`. Apr. 2016 (cited on page 26).

[64] Philip B. Stark. *Pseudo-Random Number Generator using SHA-256*. `https://www.stat.berkeley.edu/~stark/Java/Html/sha256Rand.htm`. Jan. 2017 (cited on page 26).

[65] Donald E. Eastlake 3rd. *Publicly Verifiable Nominations Committee (NomCom) Random Selection*. RFC 3797. June 2004 (cited on page 26).

[66] Vassil Roussev and Candice Quates. *the sdhash tutorial*. `http://roussev.net/sdhash/tutorial/03-quick.html`. Aug. 2013 (cited on page 27).

[67] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1996 (cited on page 29).

[68] Ronald A Fisher and Frank Yates. *Statistical Tables for Biological, Agricultural and Medical Research*. London: Oliver and Boyd, 1943 (cited on page 29).

[69] Alexei Bulazel and Bülent Yener. ‘A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web’. In: *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*. Vienna, Austria: ACM, 2017, 2:1–2:21 (cited on page 33).

[70] Jim Yuill et al. ‘Honeyfiles: Deceptive Files for Intrusion Detection’. In: *Proc. of the IEEE Work. on Information Assurance, United States Military Academy, West Point, NY*. 2004 (cited on page 33).

[71] Brian M Bowen et al. ‘Baiting inside attackers using decoy documents’. In: *International Conference on Security and Privacy in Communication Systems*. Springer. 2009, pp. 51–70 (cited on pages 34, 36).

[72] Ari Juels and Ronald L. Rivest. ‘Honeywords: Making Password-cracking Detectable’. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*. CCS ’13. Berlin, Germany: ACM, 2013, pp. 145–160 (cited on page 34).

[73] D. Balfanz et al. ‘In search of usable security: five lessons from the field’. In: *IEEE Security Privacy* 2.5 (Sept. 2004), pp. 19–24 (cited on page 36).

[74] J.A. Gómez-Hernández, L. Álvarez-González, and P. García-Teodoro. ‘R-Locker: Thwarting ransomware action through a honeyfile-based approach’. In: *Computers & Security* 73 (2018), pp. 389–398 (cited on pages 38, 40, 43).

[75] C. Moore. ‘Detecting Ransomware with Honeypot Techniques’. In: *2016 Cybersecurity and Cyberforensics Conference (CCC)*. Aug. 2016, pp. 77–81 (cited on page 38).

[76] Routa Moussaileb et al. ‘Ransomware’s Early Mitigation Mechanisms’. In: *Proceedings of the 13th International Conference on Availability, Reliability and Security*. ARES 2018. Hamburg, Germany: ACM, 2018, 2:1–2:10. ISBN: 978-1-4503-6448-5 (cited on page 38).

[77] Yun Feng, Chaoge Liu, and Baoxu Liu. ‘Poster: A New Approach to Detecting Ransomware with Deception’. In: *38th IEEE Symposium on Security and Privacy Workshops*. 2017 (cited on page 39).

[78] Mark E Russinovich, David A Solomon, and Alex Ionescu. *Windows internals*. Pearson Education, 2012 (cited on page 41).

[79] N. C. Rowe. ‘Measuring the Effectiveness of Honeypot Counter-Counterdeception’. In: *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS’06)*. Vol. 6. Jan. 2006, pp. 129c–129c (cited on page 41).

[80] Galen Hunt and Doug Brubacher. ‘Detours: Binary Interception of Win32 Functions’. In: *Proceedings of the 3rd Conference on USENIX Windows NT Symposium - Volume 3*. WINSYM’99. Seattle, Washington: USENIX Association, 1999, pp. 14–14 (cited on page 44).

[81] Mohammed I. Al-Saleh and Jedidiah R. Crandall. ‘Application-level Reconnaissance: Timing Channel Attacks Against Antivirus Software’. In: *Proceedings of the 4th USENIX Conference on Large-scale Exploits and Emergent Threats*. LEET’11. Boston, MA: USENIX Association, 2011 (cited on pages 49, 68).

[82] Ilsun You and Kangbin Yim. ‘Malware Obfuscation Techniques: A Brief Survey’. In: *International Conference on Broadband, Wireless Computing, Communication and Applications*. BWCCA ’10. Fukuoka, Japan: IEEE, 2010 (cited on page 49).

[83] Hyrum S. Anderson et al. *Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning*. 2018. arXiv: `1801.08917 [cs.CR]` (cited on page 49).

[84] Feng Xue. *Attacking Antivirus*. 2008. URL: `https://blackhat.com/presentations/bh-europe-08/Feng-Xue/Presentation/bh-eu-08-xue.pdf` (cited on pages 49, 68).

[85] Joxean Koret and Elias Bachaalany. *The Antivirus Hacker’s Handbook*. Indianapolis, IN, USA: John Wiley & Sons, 2015 (cited on page 49).

[86] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. ‘Crowdroid: Behavior-based Malware Detection System for Android’. In: *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM ’11. Chicago, Illinois, USA: ACM, 2011, pp. 15–26. ISBN: 978-1-4503-1000-0. DOI: `10.1145/2046614.2046619`. URL: `http://doi.acm.org/10.1145/2046614.2046619` (cited on page 49).

[87] Heng Yin et al. ‘Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis’. In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. CCS ’07. Alexandria, Virginia, USA: ACM, 2007, pp. 116–127. ISBN: 978-1-59593-703-2. DOI: `10.1145/1315245.1315261`. URL: `http://doi.acm.org/10.1145/1315245.1315261` (cited on page 49).

[88] Mohsen Ahmadvand, Alexander Pretschner, and Florian Kelbert. ‘A taxonomy of software integrity protection techniques’. In: *Advances in Computers*. Vol. 112. Cambridge, MA, USA: Elsevier, 2019, pp. 413–486 (cited on page 49).

[89] Battista Biggio and Fabio Roli. ‘Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning’. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’18. Toronto, Canada: ACM, 2018, pp. 2154–2156. ISBN: 978-1-4503-5693-0. DOI: `10.1145/3243734.3264418`. URL: `http://doi.acm.org/10.1145/3243734.3264418` (cited on page 49).

[90] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. ‘Making Machine Learning Robust Against Adversarial Inputs’. In: *Commun. ACM* 61.7 (June 2018), pp. 56–66. ISSN: 0001-0782. DOI: `10.1145/3134599`. URL: `http://doi.acm.org/10.1145/3134599` (cited on page 49).

[91] Dhilung Kirat and Giovanni Vigna. ‘MalGene: Automatic Extraction of Malware Analysis Evasion Signature’. In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS ’15. Denver, Colorado, USA: ACM, 2015, pp. 769–780 (cited on page 49).

[92] S. Josefsson. *The Base16, Base32, and Base64 Data Encodings*. RFC 4648. http://www.rfc-editor.org/rfc/rfc4648.txt. RFC Editor, Oct. 2006. URL: http://www.rfc-editor.org/rfc/rfc4648.txt (cited on page 53).

[93] TaxSlayer Pro. *Quick Start Manual*. 2017. URL: http://downloads.taxslayer.com/online/2017-Quick-Start-Manual.pdf (cited on page 56).

[94] IT Services of Mitchell Hamline School of Law. *Technology Notice – Disable Antivirus before using Examplify*. Dec. 2017. URL: https://mitchellhamline.edu/technology/2017/12/03/technology-notice-disable-antivirus-before-using-examplify/ (cited on page 56).

[95] AV-TEST. *The best antivirus software for Windows Home User*. 2020. URL: https://www.av-test.org/en/antivirus/home-windows/windows-10/february-2020/ (cited on page 59).

[96] AV-Comparatives. *Malware Protection Test March 2020*. 2020. URL: https://www.av-comparatives.org/tests/malware-protection-test-march-2020/ (cited on page 59).

[97] OPSWAT. *Windows Anti-Malware Market Share Report*. 2020. URL: https://www.opswat.com/blog/windows-anti-malware-market-share-report-april-2020 (cited on page 63).

[98] Luis von Ahn et al. 'CAPTCHA: Using Hard AI Problems for Security'. In: *Advances in Cryptology — EUROCRYPT 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 294–311 (cited on page 65).

[99] Guixin Ye et al. 'Using Generative Adversarial Networks to Break and Protect Text Captchas'. In: *ACM Trans. Priv. Secur.* 23.2 (Apr. 2020) (cited on page 65).

[100] Marti Motoyama et al. 'Re: CAPTCHAs: Understanding CAPTCHA-Solving Services in an Economic Context'. In: *Proceedings of the 19th USENIX Conference on Security*. USENIX Security'10. Washington, DC: USENIX Association, 2010, p. 28 (cited on page 66).

[101] Drew Springall et al. 'Security Analysis of the Estonian Internet Voting System'. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: ACM, 2014, pp. 703–715. ISBN: 978-1-4503-2957-6 (cited on page 66).

[102] S. Maruyama, S. Wakabayashi, and T. Mori. 'Tap 'n Ghost: A Compilation of Novel Attack Techniques against Smartphone Touchscreens'. In: *2019 2019 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2019, pp. 628–645 (cited on page 66).

[103] Kenneth C. Wilbur and Yi Zhu. 'Click Fraud'. In: *Marketing Science* 28.2 (2009), pp. 293–308 (cited on page 66).

[104] NIST. *NVD – CVE-2017-7150*. Oct. 2017. URL: https://nvd.nist.gov/vuln/detail/CVE-2017-7150 (cited on pages 66, 67).

[105] Andy Greenberg. *Another Mac Bug Lets Hackers Invisibly Click Security Prompts*. 2019. URL: https://www.wired.com/story/apple-macos-bug-synthetic-clicks/ (cited on page 66).

[106] Karsten Nohl, Sascha Krißler, and Jakob Lell. *BadUSB—On accessories that turn evil*. July 2014. URL: https://srlabs.de/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf (cited on page 67).

[107] Chris Paget (alias Foon). *Exploiting design flaws in the Win32 API for privilege escalation*. Aug. 2002. URL: https://web.archive.org/web/20060904080018/http://security.tombom.co.uk/shatter.html (cited on page 67).

[108] Mihai Christodorescu and Somesh Jha. 'Testing malware detectors'. In: *ACM SIGSOFT Software Engineering Notes* 29.4 (2004), pp. 34–44 (cited on page 68).

[109] Monirul I. Sharif et al. *Impeding Malware Analysis Using Conditional Code Obfuscation*. San Diego, CA, USA, Feb. 2008 (cited on page 68).

[110] Joxean Koret. *Breaking Antivirus Software*. 2014. URL: `http://joxeankoret.com/download/breaking_av_software_44con.pdf` (cited on page 68).

[111] Joxean Koret. *AV: Additional Vulnerabilities*. 2016. URL: `https://www.hoystreaming.com/wp-content/uploads/2016/03/hb_bilbo.pdf` (cited on page 68).

[112] Tavis Ormandy. *How to Compromise the Enterprise Endpoint*. 2016. URL: `https://googleprojectzero.blogspot.com/2016/06/how-to-compromise-enterprise-endpoint.html` (cited on page 68).

[113] Tavis Ormandy. *Analysis and Exploitation of an ESET Vulnerability*. 2015. URL: `https://googleprojectzero.blogspot.com/2015/06/analysis-and-exploitation-of-eset.html` (cited on page 68).

[114] Microsoft. *Driver security checklist*. 2019. URL: `https://docs.microsoft.com/en-us/windows-hardware/drivers/driversecurity/driver-security-checklist` (cited on page 69).

[115] D. E. Bell and L. J. La Padula. *Secure computer system: Unified exposition and Multics interpretation*. Technical Report ESD-TR-75-306. Mitre Corporation, Mar. 1976 (cited on page 69).

[116] John Rushby. *The Bell and La Padula Security Model*. Draft Technical Note. Computer Science Laboratory, SRI International. Menlo Park, CA, June 1986 (cited on page 69).

[117] AV-TEST. *Malware Statistics & Trends Report*. `https://www.av-test.org/en/statistics/malware/`. 2019 (cited on page 71).

[118] Cisco. *Cisco 2018 Annual Cybersecurity Report*. `https://www.cisco.com/c/dam/m/hu_hu/campaigns/security-hub/pdf/acr-2018.pdf`. Feb. 2018 (cited on page 71).

[119] Minerva Labs. *2017 – The Year Malware Went More Evasive*. `https://l.minerva-labs.com/hubfs/Minerva%20Infographic_Final.pdf`. 2017 (cited on page 71).

[120] MalwareTech. *How to Accidentally Stop a Global Cyber Attacks*. `https://www.malwaretech.com/2017/05/how-to-accidentally-stop-a-global-cyber-attacks.html`. 2017 (cited on page 71).

[121] Jonathan A. P. Marpaung, Mangal Sain, and Hoon-Jae Lee. 'Survey on malware evasion techniques: State of the art and challenges'. In: *14th International Conference on Advanced Communication Technology (ICACT)*. Piscataway, New Jersey, US: IEEE, Feb. 2012, pp. 744–749 (cited on page 71).

[122] Davide Balzarotti et al. 'Efficient detection of split personalities in malware'. In: *17th Annual Network and Distributed System Security Symposium (NDSS 2010)*. San Diego, US: The Internet Society, Feb. 2010 (cited on page 71).

[123] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. 'BareCloud: Bare-metal Analysis-based Evasive Malware Detection'. In: *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pp. 287–301. URL: `https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kirat` (cited on page 71).

[124] Dhilung Kirat and Giovanni Vigna. 'MalGene: Automatic Extraction of Malware Analysis Evasion Signature'. In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Denver, Colorado, USA: ACM, 2015, pp. 769–780 (cited on page 72).

[125] Cuckoo Sandbox. *Cuckoo Sandbox – Automated Malware Analysis*. https://cuckoosandbox.org/. 2019 (cited on page 72).

[126] Olivier Ferrand. 'How to detect the cuckoo sandbox and to strengthen it?' In: *Journal of Computer Virology and Hacking Techniques* 11.1 (2015), pp. 51–58 (cited on page 72).

[127] VirusTotal. *VirusTotal Threat Intelligence*. https://www.virustotal.com/. 2019 (cited on pages 72, 114).

[128] Marcos Sebastián et al. 'Avclass: A tool for massive malware labeling'. In: *International Symposium on Research in Attacks, Intrusions, and Defenses*. Cham: Springer International Publishing, 2016, pp. 230–253 (cited on pages 73, 89, 103).

[129] Akira Yokoyama et al. 'SandPrint: Fingerprinting Malware Sandboxes to Provide Intelligence for Sandbox Evasion'. In: *Research in Attacks, Intrusions, and Defenses*. Cham: Springer International Publishing, 2016, pp. 165–187 (cited on page 75).

[130] Scott Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. BCP 14. http://www.rfc-editor.org/rfc/rfc2119.txt. RFC Editor, Mar. 1997. URL: http://www.rfc-editor.org/rfc/rfc2119.txt (cited on page 81).

[131] Y. Dodis et al. 'On the (im)possibility of cryptography with imperfect randomness'. In: *45th Annual IEEE Symposium on Foundations of Computer Science*. Oct. 2004, pp. 196–205 (cited on page 82).

[132] Mihir Bellare et al. 'Hedged Public-Key Encryption: How to Protect against Bad Randomness.' In: *Asiacrypt*. Vol. 5912. Springer, pp. 232–249 (cited on page 82).

[133] Luciano Bello. *Debian Security Advisory: DSA-1571-1 openssl – predictable random number generator*. May 2008. URL: http://www.debian.org/security/2008/dsa-1571 (cited on page 83).

[134] *Juniper Networks: Out of Cycle Security Bulletin*. retrieved July 17, 2017 from https://kb.juniper.net/InfoCenter/index?page=content&id=JSA10713. Dec. 2015 (cited on page 83).

[135] Derek Soeder, Christopher Abad, and Gabriel Acevedo. *Black-Box Assessment of Pseudorandom Algorithms*. Black Hat USA, https://media.blackhat.com/us-13/US-13-Soeder-Black-Box-Assessment-of-Pseudorandom-Algorithms-WP.pdf. 2013 (cited on page 83).

[136] Michael Howard and David Le Blanc. *Writing Secure Code*. 2nd ed. Developer Best Practices. Microsoft Press, Dec. 2004 (cited on page 83).

[137] Microsoft Corporation. *Windows Authenticode Portable Executable Signature Format*. Tech. rep. http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/Authenticode_PE.docx. Mar. 2008. URL: http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/Authenticode_PE.docx (cited on page 86).

[138] Microsoft. *Working with the AppInit_DLLs registry value*. Nov. 2006. URL: https://support.microsoft.com/en-us/help/197571/working-with-the-appinit-dlls-registry-value (cited on pages 87, 93).

[139] Thomas M. Chen and Saeed Abu-Nimeh. 'Lessons from stuxnet'. In: *Computer* 44.4 (2011), pp. 91–93 (cited on pages 94, 109).

[140] Peter Szor. *Duqu–Threat Research and Analysis*. Nov. 2011. URL: https://securingtomorrow.mcafee.com/wp-content/uploads/2011/10/Duqu.pdf (cited on pages 94, 109).

[141] VirusTotal. *Scan report*. June 2017. URL: https://virustotal.com/en/file/81fdbf04f3d0d9a85e0fbb092e257a2dda14c5d783f1c8bf3bc41038e0a78688/analysis/ (cited on pages 94, 109).

[142] Microsoft. *Named Pipes*. Retrieved June 3, 2019 from https://docs.microsoft.com/en-us/windows/desktop/ipc/named-pipes. May 2018 (cited on page 98).

[143] Thanh Bui et al. ‘Man-in-the-Machine: Exploiting Ill-Secured Communication Inside the Computer’. In: *27th USENIX Security Symposium*. Baltimore, MD: USENIX Association, 2018, pp. 1511–1525 (cited on page 98).

[144] KnowBe4. *KnowBe4 Alert: New Strain Of Sleeper Ransomware*. Retrieved June 1, 2019 from https://www.knowbe4.com/press/knowbe4-alert-new-strain-of-sleeper-ransomware. May 2015 (cited on page 100).

[145] Avast. *PC Trends Report 2019*. Retrieved June 1, 2019 from https://blog.avast.com/pc-trends-reports. Apr. 2019 (cited on pages 102, 103).

[146] H. Cormac. ‘So long, and no thanks for the externalities: the rational rejection of security advice by users’. In: *Proc. of the 2009 New Security Paradigm Workshop (NSPW), September 8–11, 2009, Oxford, United Kingdom*. ACM, 2009, pp. 133–144 (cited on page 105).

[147] Michael Young and Ryan Zisk. *Decrypting the NegozI Ransomware*. Retrieved June 1, 2019 from https://yrz.io/decrypting-the-negozi-ransomware. 2017 (cited on page 105).

[148] Greg Hoglund and James Butler. *Rootkits: subverting the Windows kernel*. Addison-Wesley Professional, 2006 (cited on page 109).

[149] Spencer Smith and John Harrison. *Rootkits*. 2012. URL: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/rootkits.pdf (cited on page 109).

[150] Microsoft. *Kernel patch protection: frequently asked questions*. Jan. 2007. URL: https://msdn.microsoft.com/en-us/library/windows/hardware/Dn613955(v=vs.85).aspx (cited on page 109).

[151] Christian Collberg, Clark Thomborson, and Douglas Low. ‘Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs’. In: *Proc. 25th ACM Symp. Principles of Programming Languages*. POPL ’98. California, USA, 1998 (cited on page 110).

[152] Ilsun You and Kangbin Yim. ‘Malware Obfuscation Techniques: A Brief Survey’. In: *Proc. 5th Int. Conf. Broadband, Wireless Computing, Commun. and Applicat.* BWCCA ’10. Fukuoka, Japan, 2010 (cited on page 110).

[153] Arini Balakrishnan and Chloe Schulze. *Code obfuscation literature survey*. 2005 (cited on page 110).

[154] Jean-Marie Borello and Ludovic Mé. ‘Code obfuscation techniques for metamorphic viruses’. In: *Journal in Computer Virology* 4.3 (2008), pp. 211–220 (cited on page 110).

[155] Xabier Ugarte-Pedrero et al. ‘SoK: deep packer inspection: a longitudinal study of the complexity of run-time packers’. In: *Proc. 36th IEEE Symp. on Security and Privacy*. S&P ’15. California, USA, 2015 (cited on page 110).

[156] Stanley Chow et al. ‘White-Box Cryptography and an AES Implementation’. In: *Selected Areas in Cryptography*. SAC ’02. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 250–270 (cited on page 111).

[157] Stanley Chow et al. ‘A White-Box DES Implementation for DRM Applications’. In: *Proc. ACM Workshop on Digital Rights Manage.* DRM ’02. Washington, USA, 2003 (cited on page 111).

[158] Andrey Bogdanov and Takanori Isobe. ‘White-Box Cryptography Revisited: Space-Hard Ciphers’. In: *Proc. 22nd ACM Conf. Comput. and Commun. Security*. CCS ’15. Denver, USA, 2015 (cited on page 111).

[159] Marc Beunardeau et al. ‘White-box cryptography: Security in an insecure environment’. In: *IEEE Security & Privacy* 14.5 (2016), pp. 88–92 (cited on page 111).

[160] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. 'Towards Practical White-box Cryptography: Optimizing Efficiency and Space Hardness'. In: *Proc. 22nd Int. Conf. Theory and Application of Cryptology and Inform. Security.* ASIACRYPT '16. Hanoi, Vietnam, 2016 (cited on page 111).

[161] Yin Jia, TingTing Lin, and Xuejia Lai. 'A generic attack against white box implementation of block ciphers'. In: *Proc. Int. Conf. Comput. Inform. and Telecommun. Systems.* CITS '16. Kunming, China, 2016 (cited on page 111).

[162] Ondrej Kubovič. *Ransomware is everywhere, but even black hats make mistakes.* Apr. 2016. URL: https://www.welivesecurity.com/2016/04/28/ransomware-is-everywhere-but-even-black-hats-make-mistakes/ (cited on page 111).

[163] Jayavardhana Gubbi et al. 'Internet of Things (IoT): A vision, architectural elements, and future directions'. In: *Future generation computer systems* 29.7 (2013), pp. 1645–1660 (cited on page 111).

[164] Kai Zhao and Lina Ge. 'A Survey on the Internet of Things Security'. In: *Proc. 9th Int. Conf. Computational Intelligence and Security.* CIS '13. Leshan, China, 2013 (cited on page 111).

[165] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 'The Internet of Things: A survey'. In: *Computer Networks* 54.15 (2010), pp. 2787–2805 (cited on page 111).

[166] Elizabeth Weise. *Ransomware attack hit San Francisco train system.* Nov. 2016. URL: https://www.usatoday.com/story/tech/news/2016/11/28/san-francisco-metro-hack-meant-free-rides-saturday/94545998/ (cited on page 111).

[167] Dan Bilefsky. *Hackers Use New Tactic at Austrian Hotel: Locking the Doors.* Jan. 2017. URL: https://www.nytimes.com/2017/01/30/world/europe/hotel-austria-bitcoin-ransom.html (cited on page 112).

[168] Andrew Tierney. *Thermostat Ransomware: a lesson in IoT security.* Aug. 2016. URL: https://www.pentestpartners.com/security-blog/thermostat-ransomware-a-lesson-in-iot-security/ (cited on page 112).

[169] Radwire. *"BrickerBot" Results In PDoS Attack.* May 2017. URL: https://security.radware.com/ddos-threats-attacks/brickerbot-pdos-permanent-denial-of-service/ (cited on page 112).

[170] Andy Greenberg. *Hackers Remotely Kill a Jeep on the Highway—With Me in It.* July 2015. URL: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/ (cited on page 112).

[171] Paul Wagenseil. *75 Percent of Bluetooth Smart Locks Can Be Hacked.* Aug. 2016. URL: http://www.tomsguide.com/us/bluetooth-lock-hacks-defcon2016,news-23129.html (cited on page 112).

[172] Bruce Schneier. *The Internet of Things Is Wildly Insecure – And Often Unpatchable.* Jan. 2014. URL: https://www.wired.com/2014/01/theres-no-good-way-to-patch-the-internet-of-things-and-thats-a-huge-problem/ (cited on page 112).

[173] Paul Ducklin. *Popcorn Time ransomware lets you off if you infect two other people.* Dec. 2016. URL: https://nakedsecurity.sophos.com/2016/12/15/popcorn-time-ransomware-lets-you-off-if-you-infect-two-other-people/ (cited on page 112).

[174] Vindu Goel. *Verizon Will Pay $350 Million Less for Yahoo.* Feb. 2017. URL: https://www.nytimes.com/2017/02/21/technology/verizon-will-pay-350-million-less-for-yahoo.html (cited on page 112).

[175] Amanda Holpuch. *Sony email hack: what we've learned about greed, racism and sexism.* Dec. 2014. URL: https://www.theguardian.com/technology/2014/dec/14/sony-pictures-email-hack-greed-racism-sexism (cited on page 113).

[176] F. Fischer et al. ‘Stack Overflow Considered Harmful? The Impact of Copy Paste on Android Application Security’. In: *2017 IEEE SP*. May 2017, pp. 121–136 (cited on page 113).

[177] *Free Automated Malware Analysis Service*. https://www.hybrid-analysis.com/. 2019 (cited on page 114).

[178] *.NET debugger and assembly editor*. https://github.com/0xd4d/dnSpy. 2019 (cited on page 114).

[179] Utku Şen. *HiddenTear: an open source ransomware-like file crypter kit*. https://github.com/utkusen/hidden-tear. 2015 (cited on page 115).

[180] Thad Van den Bosch. *Encrypt/Decrypt Files in VB.NET (Using Rijndael)*. https://www.codeproject.com/Articles/12092/Encrypt-Decrypt-Files-in-VB\-NET-Using-Rijndael. Nov. 2005 (cited on pages 115–117).

[181] Steve Lydford. *File Encryption and Decryption in C#*. https://www.codeproject.com/Articles/26085/File-Encryption-and-Decryption-in-C. May 2008 (cited on pages 115, 116).

[182] Matt Johnson. *How do you do Impersonation in .NET? (rev. 2)*. https://stackoverflow.com/revisions/7250145/2. Sept. 2008 (cited on pages 115, 116).

[183] Utku Şen. *Destroying The Encryption of Hidden Tear Ransomware*. https://utkusen.com/blog/destroying\-the-encryption-of-hidden-tear-ransomware.html. Nov. 2015 (cited on pages 115, 118).

[184] Jornt van der Wiel. *Hidden tear and its spin offs*. Kaspersky Lab. Feb. 2016 (cited on pages 115, 118).

[185] Y. Acar et al. ‘How Internet Resources Might Be Helping You Develop Faster but Less Securely’. In: *IEEE Security and Privacy* 15.2 (2017), pp. 50–60 (cited on page 117).

[186] Burt Kaliski. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. RFC 2898. RFC Editor, Sept. 2000, pp. 1–12. URL: https://www.rfc-editor.org/rfc/rfc2898.txt (cited on page 118).

[187] Trey Herr and Paul Rosenzweig. ‘Cyber Weapons & Export Control: Incorporating Dual Use with the PrEP Model’. In: *J. National Security Law and Policy* 8.2 (2015) (cited on pages 117, 120).

[188] Phillip Rogaway. ‘The Moral Character of Cryptographic Work’. In: Austin, TX: USENIX Association, 2016 (cited on page 117).

[189] Eduard Kovacs. *Educational Ransomware Abused by Cybercriminals*. https://www.securityweek.com\/educational\-ransomware-abused-cybercriminals. Jan. 2016 (cited on page 118).

[190] Christiaan Beek. *In June, almost 30 percent of the ’new’ ransomware species we discovered was based on the HiddenTear code*. https://twitter.com/ChristiaanBeek/status/899557658071633920. Aug. 2017 (cited on page 118).

[191] *Ransomware Recap: The Ongoing Development of Hidden Tear Variants*. Trend Micro Inc. June 2017 (cited on page 118).

[192] Willy Fox. *Arescrypt: Experimental ransomware for Windows 7+ with AES-256 support*. https://github.com/BlackVikingPro/arescrypt. [20-Jul-2019]. 2017 (cited on page 119).

[193] Tarcísio Marinho. *GonnaCry: A Linux Ransomware*. https://github.com/tarcisio-marinho/GonnaCry. [20-Jul-2019]. 2017 (cited on page 119).

[194] Ivan Šincek. *Ransomware: PHP ransomware that encrypts your files as well as file and directory names*. https://github.com/ivan-sincek/ransomware. [20-Jul-2019]. 2019 (cited on page 119).

[195] Mauri de Souza Nunes. *Ransomware: A POC Windows crypto-ransomware (Academic)*. https://github.com/mauri870/ransomware. [20-Jul-2019]. 2016 (cited on page 119).

[196] Angelo Rosa. *HiddenTear (forked)*. https://github.com/Virgula0/hidden-tear. [20-Jul-2019]. 2017 (cited on page 119).

[197] Maksym Zaitsev. *CryptoTrooper: The world's first Linux white-box ransomware*. https://github.com/cryptolok/CryptoTrooper. [20-Jul-2019]. 2016 (cited on page 120).

[198] Catalin Cimpanu. *New Open Source Linux Ransomware Shows Infosec Community Divide*. Softpedia. Sept. 2016 (cited on page 120).

[199] Alex P. Schmid. 'The Definition of Terrorism'. In: *The Routledge Handbook of Terrorism Research*. Oxon, UK: Routledge, 2011. Chap. 2, pp. 39–157 (cited on page 120).

[200] Mark M. Lowenthal. *Intelligence: From Secrets to Policy*. 7th ed. Los Angeles: CQ Press, 2016 (cited on page 121).

[201] Petrus Duvenage, Sebastian von Solms, and Manuel Corregedor. 'The Cyber Counterintelligence Process: A Conceptual Overview and Theoretical Proposition'. In: *Proc. of the 14th ECCWS*. ACPI, 2015, pp. 42–52 (cited on page 121).

[202] Kevin Coleman. *Counter Cyber Intelligence*. https://www.military.com/defensetech/2009/03/09/counter-cyber-intelligence. Mar. 2009 (cited on page 121).

[203] Danny Yuxing Huang et al. 'Tracking ransomware end-to-end'. In: *IEEE Security and Privacy*. IEEE. 2018, pp. 618–631 (cited on page 122).

[204] Webroot. *2018 Webroot Threat Report Mid-Year Update*. Tech. rep. Webroot Inc., Sept. 2018. URL: https://www.webroot.com/download_file/2780 (cited on page 125).

[205] Statcounter GlobalStats. *Desktop Operating System Market Share Worldwide*. 2020. URL: https://gs.statcounter.com/os-market-share/desktop/worldwide (visited on 09/04/2020) (cited on page 125).