

# Evaluation of Live Forensic Techniques, Towards Current Cryptographic Ransomware Mitigation.

Luis Fernandez de Loaysa y Babiano

Submitted in partial fulfilment of the requirements of  
Edinburgh Napier University for the Degree of  
Cybersecurity and Forensics

School of Computing

April 2022

## Authorship Declaration

I, Luis Fernandez Loaysa Babiano confirm that this dissertation and the work presented in it are my own achievement.

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 dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed: Luis Loaysa

Date: 15/03/2022

Matriculation no: 40428544

## General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

The University may not make this dissertation available to others.

## Abstract

Ransomware has been established as one of the largest current threats to organisations, small businesses, governments, and individuals alike. The appearance of cryptocurrencies and enhancement of encryption key management schemes increased the capacity to compromise the victim's data and demand ransom payments. The variety of ransomware families and their continued evolution makes the task of preventing these attacks complex. Some ransomware try to bypass antivirus engines by using concealing techniques while others can detect a virtualised environment to hinder dynamic analysis. Current types target victims via the language of the target operating system and most now use complex multi-layer encryption techniques which cannot be mitigated using conventional methods.

Recent studies have shown that it is possible to find the decryption keys in the volatile memory of an infected machine while the ransomware is being executed, in a type of side channel attack. However, the related work in the field does not seem to have addressed the current cryptography used by ransomware, including stream ciphers especially Sala20 and the use of unique keys per victim files which is now common. This work reproduces these latest cryptographic management techniques being used, and in a virtual environment explored proof of concepts for both Salsa key extraction from memory, and one key per file ransomware encryption symmetric extraction. Additionally, the methods prototyped have been evaluated against real recent ransomware samples.

This proof of concept was shown to successfully recover Sala20 keys and nonce pairs and decrypt victim files using live memory forensics methods. This would allow victims to recover files without paying a ransom and seems to be a new method in the field of key extraction from memory. The findings from the one key per file experiments seem to show that it's possible to use live memory forensics to extract many ransomware symmetric keys, and thus could be used to mitigate the most recent cryptographic ransomware attack methods. Current ransomware samples were also experimented with to evaluate the methods, and to validate the decryption of real ransomware encrypted files. This showed successful decryption of large numbers of files was possible and adds a new method to the field of ransomware mitigation.

## Acknowledgements

I would like to express my gratitude to my supervisor Rich Macfarlane for his relentless guidance, inspiration and time spent reviewing this work. I would also like to thank Simon Davies for his expertise and dedication throughout the project. Without them this work would have not been possible.

I would also like to thank my friend Jonny for reading my dissertation on more than one occasion and to my partner Aisha for her support and love during challenging times. Last but not least to my mother that with her kindness and patience has always encouraged me.

# List of Contents

<b>1. INTRODUCTION .....</b>	<b>1</b>
1.1 Background .....	1
1.2 Aim and Objectives .....	2
1.3 Ethical Considerations .....	2
1.4 Layout of the Document .....	3
<b>2. LITERATURE REVIEW .....</b>	<b>4</b>
2.1 Introduction .....	4
2.2 Classification of Ransomware .....	4
2.2.1 Cryptographic Ransomware Key Management .....	5
2.3 Mitigation Techniques .....	7
2.3.1 Cryptographic Ransomware Detection Strategies .....	7
2.3.2 Reactive Mechanisms .....	9
2.4 Mitigation and Recovery Using Live Forensics Techniques .....	12
2.5 Cryptographic Ransomware and Associated Research .....	14
2.5.1 WannaCry .....	15
2.5.2 GandCrab and Sodinokibi .....	15
2.5.2.1 Sodinokibi .....	16
2.5.3 Petya.....	16
2.5.3.1 NotPetya.....	17
2.5.4 Phobos.....	17
2.5.5 CryptoWall.....	17
2.5.6 DarkSide.....	18
2.5.7 Taxonomy and Key Schemes .....	18
2.6 Conclusion .....	19
<b>3. EXPERIMENTAL DESIGN AND IMPLEMENTATION .....</b>	<b>21</b>

---

3.1 Introduction.....	21
3.2 Methods .....	21
3.3 Experimental Design .....	23
3.3.1 Initial AES Key Extraction Experiment .....	23
3.3.2 Initial Environment Design .....	24
3.3.3 Technical Details of Initial Experiments .....	25
3.3.4 Initial Experiment Stage 1 – Synthetic Ransomware using AES.....	26
3.3.5 Initial Experiment Stage 2 – Extract Memory and AES Keys .....	27
3.3.6 Initial Experiment Stage 3 – Validate the Extracted AES Keys .....	27
3.3.7 Initial Experiments in Identifying Salsa20 Keys in Memory.....	28
3.3.7.1 Identifying Salsa20 Initial State Matrix .....	30
3.3.7.2 Salsa20 Key Extraction Tool.....	31
3.3.7.3 Initial Experiment Stage 1 – Synthetic Ransomware using Salsa20 .....	32
3.3.7.4 Initial Experiment Stage 2 – Extract Memory and Salsa20 Keys .....	33
3.3.7.5 Initial Experiment Stage 3 – Validate the Extracted Salsa20 Keys.....	33
3.4 Real Ransomware Experiments Design.....	35
3.4.1 Real Ransomware Environment Design .....	35
3.4.2 Salsa20 Ransomware Sample.....	37
3.4.3 Technical Specifications for Real Sample Experiments.....	38
3.4.4 Experiment 1 – Sodinokibi Ransomware Salsa20 Key Recovery.....	39
Stage 1 – Sodinokibi File Encryption.....	39
Stage 2 - Memory and Salsa20 Keys Extraction.....	40
Stage 3 - Validating Extracted Keys by Decryption of Files .....	42
Metadata Null Bytes Checksum Encryption Analysis .....	44
3.4.5 Experiment 1 - Evaluation .....	47
3.4.6 Experiment 2 - Sodiokibi Large Scale Key Recovery.....	47

---

3.4.8 Experiment 3 – Sodinokibi Encryption for Key Time Lining .....	51
3.5 Conclusion .....	52
<b>4. EVALUATION.....</b>	<b>54</b>
4.1 Introduction.....	54
4.2 Aims and Objectives.....	54
4.2.1 Objective One – Literature review .....	54
4.2.2 Objective Two – Design and Implementation.....	55
4.2.3 Objective Three - Evaluation .....	55
4.2 Self Appraisal.....	56
4.3 Future Work .....	57
<b>REFERENCES .....</b>	<b>58</b>
<b>APPENDIX A.....</b>	<b>66</b>
A.1 Project timeline.....	66
A.2 Diary Examples .....	67
<b>APPENDIX B.....</b>	<b>71</b>
B.1 Encrypt_AES .....	71
B.2 Decrypt_AES.....	72
B.3 Encrypt_Salsa20 .....	72
B.4 Decrypt Salsa20 .....	73
B.5 Null Bytes.....	74
B.6 Shifting Bytes .....	75
B.7 PowerShell Automation .....	76
B.8 Salsa20 Extraction Tool.....	76



## List of Figures

Figure 1 - WannaCry Encryption Scheme .....	6
Figure 2 - Initial Environment .....	24
Figure 3 - Overview Stage 1 of Initial Experiment .....	26
Figure 4 - Encrypt_AES Output .....	26
Figure 5 - File Before and After Encryption .....	26
Figure 6 - Overview Stage 2 of Initial Experiment .....	27
Figure 7 - Findaes Output.....	27
Figure 8 - Overview Stage 3 of Initial Experiment .....	27
Figure 9 - Decrypt_AES Code.....	28
Figure 10 - Decrypted Data and Original File .....	28
Figure 11 – Generated Salsa20 Keys .....	29
Figure 12 – Salsa20 Encryption Script .....	29
Figure 13 - Salsa20 being using for encryption.....	29
Figure 14 - Key Identified in Memory .....	30
Figure 15 – Initial State Matrix in Memory .....	30
Figure 16 – Initial State Matrix Diagram .....	30
Figure 17 – Pattern match the Salsa20 Initial State matrix .....	31
Figure 18 - Matrix Check .....	31
Figure 19 – Extracted Key and Nonce .....	32
Figure 20 - Overview Stage 1 of Initial Experiment with Salsa20.....	32
Figure 21 – Encrypting Multiple Files with Salsa20 Synthetic Ransomware.....	32
Figure 23 – Overview Stage 2 of Initial Experiment with Salsa20 .....	33
Figure 22 – Before and After Encryption with Salsa20 Algorithm.....	33
Figure 24 - Extracting Salsa20 and Nonce pair .....	33

---

Figure 25 - Overview Stage 3 of Initial Experiment .....	33
Figure 26 – Decrypted Data with Salsa20 Algorithm and Original File .....	34
Figure 27 - Diff Against Decrypted and Original Files .....	34
Figure 28 – Real Ransomware Environment .....	37
Figure 29 - Sodinokibi Encryption Process.....	38
Figure 30 - Overview Stage 1 – Sodinokibi File Encryption.....	39
Figure 31 - Sodinokibi Executed .....	40
Figure 32 - Files Encrypted by Sodinokibi .....	40
Figure 33 - Overview Stage 2 – Salsa20 Key Extraction.....	40
Figure 34 - Keys Extracted from Sodinokibi .....	41
Figure 35 – Salsa20 Nonce and File Related .....	41
Figure 36 – Overview Stage 3 - Key Validation .....	42
Figure 37 - Sodinokibi Appended Metadata Structure .....	42
Figure 38 – File Size Before and After Encryption .....	43
Figure 39 - Decrypting Null Bytes .....	44
Figure 40 - Encryption by Stream Ciphers.....	45
Figure 41 – Encryption by Stream Ciphers .....	45
Figure 42 – Shifting Bytes to Find Position .....	46
Figure 44 – Diff of Recovered and Original Files .....	46
Figure 43 - File Header Found .....	46
Figure 45 - Sodinokibi Decrypted and Recovered .....	47
Figure 46 – PowerShell Script.....	48
Figure 47 – Unique Keys in Memory in 10 Seconds Interval.....	49
Figure 48 – Number of Keys Extracted from Memory.....	49
Figure 49 – Persistence of Keys in Memory .....	50
Figure 50 - Modified Time After Encryption.....	51

Figure 51 - Encryption Key Timeline .....	52
Figure 52 - Project Timeline.....	66
Figure 53 - A.2.1 Diary 15/09/2021.....	67
Figure 54 - A.2.2 Diary 02/12/2021.....	68
Figure 55 - A.2.3 Diary 20/01/2022.....	69
Figure 56 - A.2.4 Diary 17/02/2022.....	70

# List of Tables

Table 1 - Mitigation Techniques Compared..... 11

Table 2 - Memory Signature of Symmetric and Asymmetric Ciphers ..... 14

Table 3 - Ransomware Taxonomy and Live Forensics as Mitigation Technique ..... 19

Table 4 – Technical Details of Initial Experiments ..... 25

Table 5 - Technical Details of Experiments ..... 38

# 1. Introduction

## 1.1 Background

Ransomware ranks high among current cyber threats around the world, with institutions such as the Interpol classifying it as a threat that requires the same international collaboration to fight terrorism, mafia groups and human trafficking (*Immediate Action Required to Avoid Ransomware Pandemic - INTERPOL*, n.d.). Ransomware attacks are not new, they are almost as old as the first computers with the first known as PC Cyborg and dating back to 1989 which targeted the World Health Organization's AIDS conference (Oz et al., 2021). With the evolution of the internet, the appearance of Bitcoin in 2009 and more robust cryptographic schemes these attacks became more aggressive over the years. CryptoLocker in 2013 and Cryptowall in 2014 added the use of TOR browser and Bitcoin to provide anonymity. The latter infected more than 600,000 systems worldwide (Oz et al., 2021). A more sombre situation came with WannaCry and NotPetya in 2017 both being able to spread to other systems autonomously and with NotPetya becoming second global security issue in the world (Humayun et al., 2021; Rehman et al., 2018).

Moreover, the COVID-19 pandemic has seen a sharp increase in the use of ransomware around the world. With more people working from home, businesses have been forced to rapidly adapt and, span their networks over larger areas, exposing them to higher risk and costing them more than double compared to the year 2020 (Beaman et al., 2021). Beaman and colleagues also point out that by the end of 2021 ransomware attacks are expected to cost the world \$20 billion, up from \$325 million in 2015. Furthermore, a new strain under the CoronaVirus name has been identified targeting Windows computers and using a phishing e-mail to trick the users to download and install the application which begins the execution of the malware to extract private and system information before encrypting the data (*CoronaVirus Ransomware*, n.d.).

Given this historical trend, the effort to mitigate ransomware attacks is far from over. Though, there are mechanisms to detect and prevent ransomware (Genç et al., 2018; Mehnaz et al., 2018), however, there are drawbacks to each of these systems (Berrueta et al., 2019; Genç, 2020). Furthermore, there seems to be a common denominator in modern ransomware involving the presence of encryption keys in the volatile memory of an infected host (Bajpai, 2020; Davies et al., 2020). The use of memory forensics techniques to retrieve encryption keys in memory can help in mitigating these attacks. However, current research does not explore if the use of these techniques can be applied to more complex ransomware to recover compromised data. For instance, recovering different keys used for different files. To bring these solutions up to date it is important to understand the ongoing struggle against ransomware and research their cryptographic

schemes. This work researches these cryptographic schemes and applies live memory forensics techniques to mitigate modern ransomware.

## 1.2 Aim and Objectives

The aim of this work is to evaluate live memory forensics techniques to extract cryptographic keys from volatile memory created by current strains of hybrid cryptographic ransomware. The focus of the work will be around current hybrid encryption ransomware and unique key per file encryption method mitigation, including exploring the current algorithms such as Salsa20.

Objective 1: To conduct a literature review around current ransomware cryptography management, ransomware detection and recovery research, and live forensic methods.

Objective 2: To design and carry out experiments to evaluate live memory forensics techniques against current ransomware strains and their encryption methods, finding multiple keys in the volatile memory of a victim machine, and trying to use these to decrypt files.

Objective 3: To evaluate and discuss the results against other related work, and an evaluation of the objectives of the project.

## 1.3 Ethical Considerations

Researching malware may raise ethical issues related to the execution, use or solutions developed. To address this concern this work complies with rigorous safeguards that treats the malware in a secure manner, preventing it from spreading or affecting any other system but the intended test files on which the analysis is carried out. This compliance is in line with the British Computing Society (BCS) code of conduct and national legislation, Computer Misuse Act 1990. The purpose of this work is to advance the research against cryptographic ransomware and therefore the malware was used only within that scope. To prevent the unintentional spread the malware was executed in a virtual machine, without exterior connections and within a physical isolated computer. This computer and its software were to date, in flight mode, without any sensitive information (such as credentials or documents) and with a disabled Network Interface Card. Lastly, when the transportation of the malware was needed, it was compressed with a protected password and without the file extension to avoid accidental execution. Upon the finalisation of the project all files and executables related to the malware were deleted and computer and transportation method formatted. These steps were deemed sufficient to prevent unintentional damage and to demonstrate adequate use of malware for analysis.

## 1.4 Layout of the Document

This document is organised as follows. The first section contains a literature review that examines the categorisation of ransomware depending on their key structure, the damage they can potentially cause, and the different detection and countermeasures techniques used against it. The literature review also examines different strains of ransomware relevant to this work and a review on current methods for key extraction used in digital forensics. The second section explains the methodology used in the experiments and the motivation behind them to solve the problem exposed in the literature review. The third section outlines the implementation of the experiments and the discussion about the results. The fourth section contains the evaluation, which compares the initial objectives to the findings of the experiments to evaluate whether they have been successful. Finally, the conclusion offers the limitations of the project and proposed future work.

## 2. Literature Review

### 2.1 Introduction

This chapter of the dissertation discusses the classification, detection, and mitigation techniques of ransomware as well as its current relation to memory forensics. Firstly, a review of the different classifications of ransomware available in the literature that include categories relevant to their key and file management is reviewed. These categories are useful to identify different strains of ransomware. Secondly, some of the most adopted mitigation techniques are described and subdivided into detection and reactive mechanisms. For instance, extracting keys from memory is considered a reactive mechanism. However, reviewing detection techniques can provide a useful insight against ransomware. Thirdly, an extensive analysis of modern types of ransoms is explored. This analysis focuses on their cryptographic schemes which is highlighted in different technical journals and is aimed to have a more practical view of the current scenery. Fourthly, the review of the overlapping fields that exist between the study and prevention of cryptographic ransomware and digital forensics is explored. Lastly, a conclusion with a summary of the different sections with the most important points derived from the literature is presented.

### 2.2 Classification of Ransomware

There is a wide variety of ransomware and ways of classifying it (Moussaileb & le Boudier, 2021; Oz et al., 2021). Arguably, the two broadest categories are in terms of malicious action which refers to what it is aimed to do in an infected host as pointed in the taxonomy survey by Oz et al. (2021). This survey also distinguishes between encrypting and locking. The former refers to cryptographic ransomware, which is newer and generally allows the user to access the system. However, its objective is to encrypt files until the user pays a ransom with the hope of recovering the data. On the other hand, locking, refers to prevent the user from accessing the device until said ransom is paid. Between these two types, Oz et al. (2021) consider cryptographic ransomware more dangerous as it is possible to bypass the locking type by re-installing the operating system without losing files.

Another relevant type of classification that focuses on cryptographic ransomware is based on its virulence, denying accessibility to the encrypted files (Zimba et al., 2021). In this classification, Zimba and colleagues propose 8 categories of ransomware based on two groups, encryption attack model and deletion attack model. The former describes the key management while the latter focuses on the volume shadow copies of the encrypted files and the rest of files that are not encrypted by the ransomware. Bajpai et al. (2018) also classify crypto ransomware, they propose 6 categories based on defects that the



implementation of the cryptographic scheme may have. For instance, if there are decryption tools available or if the same key has been used for different instances.

There is overlap between encryption attack model group described by Zimba, Wang, et al. (2021) and the 6 categories from Bajpai et al. (2018) regarding the key management. A category 8 by Zimba and colleagues consists of hybrid encryption that deletes the volume shadow copies and corrupts the remnant files while a category 1 is scareware that only deletes files that are easily recoverable. On the other hand, Bajpai and colleague's category 6 is a flawless encryption model with seemingly unretrievable data and a category 1 does not encrypt files.

These two categorisations are consistent with other literature and broader taxonomy such as that described by Al-rimy et al. (2018). In their work they address the different platforms that ransomware can infect, the target of the attack and the virulence of the infection itself. The virulence comprises scareware, locker, and crypto ransomware, in which the categories of Bajpai et al. (2018) are based. The three articles agree on that crypto ransomware is considered the most dangerous and detrimental of all since it can cause irreversible damage (Al-rimy et al., 2018; Bajpai et al., 2018; Zimba et al., 2021).

### 2.2.1 Cryptographic Ransomware Key Management

The first crypto ransomware arrived in 2013 and became the dominant form of ransomware. Ever since, crypto ransomware has been evolving and adopting more sophisticated methods to make its detection and prevention more difficult (Oz et al., 2021; Ramsdell & Esbeck, 2021).

Because of this constant presence it is useful to subcategorise crypto ransomware according to its cryptographic scheme - single or multiple key structure (Zimba et al., 2021). Crypto ransomware functionality depends on an adequate management and implementation of these schemes. For instance, a single key attack model uses a symmetric or asymmetric cipher to encrypt the victim's data. On the other hand, a multiple key structure attack uses both types at the same time.

In a multiple key structure attack, user data is encrypted using a symmetric key. This key is then encrypted by using the public key of an asymmetric key pair, the private counterpart is never used in the infected machine. It is important to note that in principle the data cannot be recovered without the private key (Bajpai, 2020; Zimba et al., 2021).

Bajpai (2020) and Oz et al (2021) found that popular symmetric and asymmetric ciphers used by crypto ransomware are Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA) respectively. They also found that symmetric ciphers have the advantage of fast encryption when compared to asymmetric. On the other hand, as Bajpai (2020) also points out, symmetric ciphers introduce a weakness since the same key is

used to encrypt and decrypt and it is exposed in memory, therefore the retrieval of the data becomes possible. Although, this window for retrieval shrinks if there is a unique key used for each file.

More recent crypto ransomware has been found to use a hybrid scheme, where asymmetric and symmetric ciphers are used (Bajpai, 2020; Genç, 2020; Zimba et al., 2021). Bajpai (2020) uses WannaCry as a case study to describe a robust cryptosystem, where symmetric keys are used to encrypt files and an asymmetric key to protect those symmetric keys. This case study is shown in Figure 1.

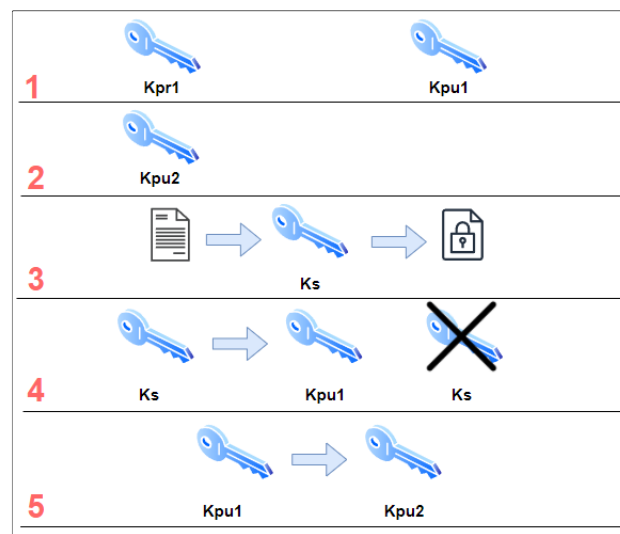


Figure 1 - WannaCry Encryption Scheme

This malware creates an asymmetric key pair in the infected machine  $K_{pr1}$  and  $K_{pu1}$ , it then proceeds to import the attacker's public key embedded in the binary  $K_{pu2}$ . After, it enumerates files of interest and encrypts them with an individual session key for each,  $K_s$ . Once the file is encrypted, WannaCry encrypts  $K_s$  with  $K_{pu1}$  and deletes the session key  $K_s$ . Finally,  $K_{pu1}$  is encrypted with  $K_{pu2}$  leaving the attacker's private key  $K_{pr2}$  as the only method to recover the data. It is important to note that  $K_{pr2}$  does not appear in the victim's computer until the ransom is paid. However,  $K_{pr1}$  and  $K_s$  might be exposed while they are in memory. Moreover, this ransomware does not need a Command and Control (C&C) to start encrypting files, meaning that internet connection is not necessary.

Lastly, hybrid cryptographic schemes are now widely adopted by ransomware (Moussaileb & le Boudier, 2021; Oz et al., 2021; Ramsdell & Esbeck, 2021). These types of attack structure are also the most damaging and difficult to recover from without paying the ransom (Bajpai, 2020; Zimba, Wang, et al., 2021). However, these classifications help to understand how crypto ransomware act and how it can be prevented or mitigated.

## 2.3 Mitigation Techniques

Crypto ransomware is irreversible without the help of the attacker when executed successfully (Moussaileb & le Boudier, 2021). This forces potential victims into using detection techniques that aim to stop the execution of the ransomware as soon as possible. However, there are also reactive techniques that focus on controlling the access to the cryptographic API on the infected host or in retrieving encryption keys. These mechanisms differ in how they treat the threat of crypto ransomware but pivot around its mitigation. The following section discusses first the different approaches on detection techniques and secondly the different reactive mechanisms.

### 2.3.1 Cryptographic Ransomware Detection Strategies

A solution against every strain of ransomware does not exist which couples with detection mechanisms becoming rapidly outdated (Moussaileb & le Boudier, 2021). However, the race against ransomware is dependent on these detection mechanisms as ransomware has been evolving since its appearance in 1989 (Oz et al., 2021). Ransomware detection techniques are varied, with 32% of the literature reviewed by Berrueta et al (2019) being machine learning. Although, as they point out, other methods use entropy measurements, file system deletion or modification among others. This detection is performed with the objective of stopping the execution of the malware and preventing extensive damage.

The encryption techniques used by crypto ransomware can be an identity signature, although it can also be confused with compressed files or benign encryption (Oz et al., 2021). An important element of encryption is that the randomness of an encrypted text is higher than that of the original data (Bajpai, 2020; Genç, 2020). This randomness is called entropy and is analogous to that described by the second law of thermodynamics, which describes that matter naturally tends to transition into a random or unordered state. This term was coined by Claude Shannon in the context of information theory (Shannon, 1948). Today it is also used to detect the levels of entropy created by encryption algorithms which is related to the presence of crypto ransomware and allow its detection independently of the strain of ransomware (Davies et al., 2021b). In this work, Davies et al (2021b) argue that there is a clear difference between the entropy at the beginning of a file with those encrypted by ransomware as opposed to the findings by Pont et al (2020) which state that statistical analysis for detecting ransomware is not reliable. It is important to note that Pont et al (2020) analyses the entirety of the files and not only the beginning which produces many false positives when comparing compressed files with encrypted files. Though this is hopeful, as it creates a new approach for accurately detecting cryptographic ransomware, Moussaileb and le Boudier (2021) note that a counter measure to this detection could be skipping the first 5,120 bytes of any file and in doing so preventing the alteration that would trigger the alert that a solution taking advantage of Davies et al (2021b) approach would have.

A different adopted detection method is machine learning. In their work, Cohen and Nissim (2018) use parameters extracted from the volatile memory to create a ransomware detection tool. They chose four real world samples of ransomware that do not require of a command and control (C&C) server to begin encryption, as they could execute it while being offline. They used Cerber, TeslaCrypt, Vipasana and Chimera. They then proceeded to extract information from 100 memory dumps of snapshots taken from infected virtual machines. Some of that extracted information included a list of thread objects using a pool scanner or kernel call-backs which were fed into a machine learning algorithm that could use this information to tell apart ransomware from other programs. While their results were very accurate, the samples of ransomware were limited. Also, as they note in their results, depending on machine learning for detecting ransomware may be a disadvantage when new ransomware with new features appears as the algorithm will not be able to discern between benign software and ransomware (Cohen & Nissim, 2018). This problem is consistent with other similar detection techniques based on signature detection to prevent the execution of this malware since malicious actors can use zero-day vulnerabilities or code obfuscation techniques to bypass the detection (Moussaileb & le Boudier, 2021).

Berrueta et al (2019) in their survey compared different ransomware detection methods, in particular one in which three indicators must be present for a machine to be classified as having ransomware. The first indicator is a quantitative comparison of original and modified files along with an entropy measurement of these new files. Incorporating Shannon entropy for detection as noted by Davies et al (2021b). The second indicator are the number of files deleted, and variety of types read by the process. The third is a risk value which increases as a process do more of the first two indicators. Once a certain threshold is reached, access to disk is blocked for the process (Berrueta et al., 2019). Although, the same survey concludes that most of these methods lack optimisation as most of them are proof-of-concept solutions and the best current protection measure is a combination of detection and backup policies. However, backups solutions are considered a drawback due to management complexity (Berrueta et al., 2019).

Another similar method based on file system changes is developed by Continella et al (2016). The technique consists of detecting ransomware by training a machine learning algorithm. This process occurs in a virtualized environment where the algorithm is instructed to differentiate between benign and malicious operations. The tool named ShieldFS, uses file system activity such as read, renamed files, entropy, or write activities as indicators to detect a malicious process. If the algorithm determines the process as malicious, the monitored changes are reverted seamlessly. Nonetheless, if the ransomware distributes its operation over several number of processes, they may be deemed benign by ShieldFS (de Gaspari et al., 2019). Importantly, this solution has an average overhead of 26% and 40% at peak performance which is a considerable disadvantage (Genç, 2020).

### 2.3.2 Reactive Mechanisms

Reactive mechanisms focus on controlling the generation of encryption keys locally. Some of these mechanisms are also referred to key escrow are based on gaining access to the encryption keys used by ransomware to recover the encrypted data (Bajpai, 2020; Genç, 2020; Oz et al., 2021).

In the work previously discussed, Berrueta et al (2019) examined the generation and recovery of cryptographic keys. They point to the No More Ransom Project which offers decryption tools for different ransomware strains. The same project offers a tool named Crypto Sheriff which aims to identify the strain of ransomware and apply the correct decryption tool. However, Filiz et al (2021) found that 49% of the strains were incorrectly identified out of 61 tried samples. Although the decryptors do work for a number of strains, the software may discourage users by misidentifying the type of ransomware and preventing the subsequent decryption.

Berrueta et al (2019) also point out two major studies from 2017 that take advantage of the ransomware need to access random numbers to generate encryption keys in an infected host. The first study proposes controlling the Pseudo Random Number Generator (PRNG) of the cryptographic API in Windows systems. This approach allows the user to control all random numbers generated by the PRNG. Therefore, the encryption keys can be retraced from these numbers to recover the encrypted files (Kim et al., 2017). The second study presents PayBreak, a software that hooks the cryptographic API to take control over the keys generated in an infected host. The authors of PayBreak noted how a malware could intentionally create false encryption keys to fill the vault where the keys are stored (Kolodenker et al., 2017). PayBreak differs from the solution of Kim et al (2017) in that all the encryption keys created using the local PRNG are stored, making it a key escrow mechanism. A problem for both proposals is that ransomware might import its own cryptographic library, with its own PRNG, instead of using that in the host machine (Bajpai, P 2020; Moussaileb & le Boudier, 2021; Oz et al., 2021). Moreover, as Genç (2020) points out in his dissertation, if PayBreak were to become ubiquitous it would present a new target for cybercriminals as all the ephemeral keys needed for other legitimate operations would be stored in the same vault. Lastly, Genç (2020) also notices that PayBreak is unable to stop NotPetya as it modifies the Master Boot Record (MBR) which makes the vault inaccessible.

After the appearance of PayBreak, Genç et al (2018) developed UshallNotpass which allows or denies access to the PRNG of the cryptographic API by applications based on their legitimacy. This means that only processes that have been whitelisted or certified in some manner can access the PRNG function. For the practical testing they set a system policy based on locally whitelisted applications so any program that does not belong in this list is denied the access to the cryptographic API. UshallNotpass is remarkably able

to mitigate NotPetya as it stops its execution before the crypto-API can be used. However, Genç (2020) notes that UshallNotpass is susceptible to a ransomware hijacking the process of a whitelisted application and be accepted as a legitimate application. For this reason he developed an improved version that unifies the interceptor and controller of the process to avoid the hijacking. This new updated version is named NoCry. While this is a modern and well founded solution, a similar problem to that of PayBreak or UshallNotpass remains unsolved, ransomware can import its own cryptographic library (Bajpai, 2020; Moussaileb & le Boudier, 2021; Oz et al., 2021). Genç (2020) argues that UshallNotpass nor NoCry were designed to stop specific types of ransomware and that these applications are to be used as complementary defence solution. Furthermore, their implementation requires either a local whitelisted database or a method to certify applications adopted by major manufacturers and developers as standard which means a gigantic world wide effort.

Another area of research is the use of decoy files which is widely used to detect and create a response against ransomware (Bajpai, 2020; Genç, 2020; Moussaileb & le Boudier, 2021). Mehnaz et al (2019) propose RWGuard, a key escrow solution that deploys decoy files in every directory and monitors processes, entropy, file type, file size and other changes in the system to detect ransomware. When a malicious activity is detected a hook in the crypto-API stores the file name and the encryption key which can be used to recover the data if the maliciousness of the process is confirmed. According to their findings, RWGuard has zero false negatives when detecting ransomware, 0.1% of false positives and an overhead of 1.9%. This strategy may face other problems such as the use of spyware which could help avoid the decoy elements (Moussaileb & le Boudier, 2021). This concern is consistent with other work which shows how malicious actors may employ lower rate of creation and modification of files as well as file attributes or path diversity to avoid file detection techniques (Shukla et al., 2015).

Another key escrow mechanism that uses decoy files is PickPocket (Bajpai, 2020). In this solution a bait file is placed strategically to trigger the process of dumping keys from the memory process that is encrypting the files. Genç (2020) argues against this type of solution as the files that were already encrypted when the mechanism is triggered would be unrecoverable. Moreover, ransomware that encrypts files with an asymmetric cipher would bypass PickPocket as the private key would never be in the system. Although, as Bajpai notes, single-key structure attacks, using asymmetric encryption are slow and resource demanding reason for which these types of ransomware are almost obsolete (Bajpai, 2020). Bajpai (2020) and Genç (2020) agree that crypto ransomware may use a symmetric cipher to create a unique key for each file encrypted and that it is difficult to capture them due to the speed in which these ciphers encrypt data. Therefore, Bajpai (2020) designs PickPocket to extract asymmetric keys and those AES keys that are found to be in the same space in memory using their key schedule.

Table 1 shows a comparison of the most recent mitigation techniques tools used against crypto ransomware found in the available literature. These tools are mostly proof-of-concept, with exception of No More Ransom which has decryptors for real ransomware. This means that they were tried in controlled laboratory settings in which the execution of the malware is expected as opposed to execution happening without the user knowledge or with zero-day vulnerabilities involved.

Table 1 - Mitigation Techniques Compared

Name	Published	Method	Efficacy	Number of samples
<b>ShieldFS</b>	Continella, 2016	File protection	<b>100%</b> encryption prevention after execution	<b>383</b> of which <b>305</b> were new (For the ML algorithm).
<b>No More Ransom</b>	From 2016 and still running (Europol and others)	Online decryptor (CryptoSheriff)	<b>43%</b> CryptoSheriff as opposed to decryptors found in the website	<b>61</b> – Against encrypted files in the wild.
<b>PayBreak</b>	Kolodenker, 2017	Key escrow	Decrypted files for <b>12</b> of the samples	<b>20</b> – Real world samples
<b>UshallNotPass</b>	Genç, 2018	Cryptographic API access	<b>94%</b> encryption prevention after execution	<b>524</b> – Real world samples
<b>PickPocket</b>	BajPai, 2020	Key escrow	Decrypted files for <b>9</b>	<b>10</b> – One of which is synthetic ransomware
<b>NoCry (Improved version of UshallNotPass)</b>	Genç, 2020	Cryptographic API access	<b>97.1%</b> encryption prevention after execution	<b>747</b> – Real world samples

Furthermore, the pass of time affects how these tools may be implemented or how effective they are against crypto ransomware. This is due to the ever evolving nature of ransomware (Moussaileb & le Boudier, 2021; Oz et al., 2021). For instance, ShieldFS in 2016 focused on monitoring files but later, there seems to be a shift in the solutions proposed with key escrow and cryptographic API control leading the way. The reason for this shift may be related to the success these tools. However, if these solution were to become the norm, a possible circumvention would be using a statically linked cryptographic library (Bajpai, 2020; Moussaileb & le Boudier, 2021; Oz et al., 2021). The

efficacy of these tools may increase or decrease compared to modern ransomware depending on their newly adopted behaviours.

There is a wealth of literature on how to detect ransomware but these methods are not always up to date with the current problem. Reactive techniques, on the other hand, always pose a risk as they depend on ransomware having certain known behaviours to be effective. Although, they could potentially allow a full recovery of the data without the need of backups or other defence mechanisms that are resource demanding. A solution that combines up to date antivirus and backups seems to be the current functional trend. However, it does not protect against infections through zero-day vulnerabilities and novel ransomware behaviour (Moussaileb & le Boudier, 2021).

Lastly, there are open questions in the literature relating to key escrow techniques. There does not seem to be available literature about extracting keys from memory to mitigate ransomware apart from Yuste and Pastrana, Bajpai and Davies et al (Bajpai, 2020; Davies et al., 2020; Yuste & Pastrana, 2021). Furthermore, these works do not address the issue of retrieving unique keys per file from memory.

## 2.4 Mitigation and Recovery Using Live Forensics Techniques

Encryption techniques are fundamental in ensuring privacy and security across different domains in computing. They have been evolving alongside other technologies such as the internet, digital banking, instant messaging, and data storage. However, they can easily be misused. One clear example of this is ransomware. In another instance, using encryption to conceal evidence in criminal cases makes the search of digital artifacts harder. This chapter reviews the potential use of memory forensics to mitigate modern crypto ransomware.

Several studies on digital forensics focus on memory key extraction. Adi Shamir and Van Someren (1999) describe different potential methods in which an attacker could retrieve key material from the memory or disk of a computer, since this information may be stored in different places, such as the swap file, kept by a program that uses a bad cryptographic design or simply stored in a file by an overconfident user. Memory forensics studies present a method to locate encryption keys which consists in calculating the Shannon entropy for an appropriate length of byte. However, Halderman et al (2009) argue that this method alone produces a lot of false positives when there is compressed files in the system and, therefore, is not reliable.

Kaplan and Geiger (2007) developed Disk Decryptor to extract key material from the volatile memory of a computer with the aim to decrypt a suspect's hard disk. Additionally, they contemplated the use of TrueCrypt as a case study. While Kaplan and Geiger (2007) did not disclose their methodology, their findings seem plausible given that capturing RSA keys from memory dumps using the key signature was previously



demonstrated (Klein, 2006). These studies are over ten years old, and given the pace of innovation in technology, this could render a solution futile. In this case, through his website, updated in 2020, Klein shows how the United States National Security Agency (NSA) used his method to extract RSA keys in the wild in 2016 (*All Your Private Keys Are Belong to Us*, n.d.). Klein's website also points to other approaches using the same technique exist today; for example, a volatility plugin that searches for signatures. Furthermore, Nakano et al (2014) show a statistical key extraction method that can be used against RSA, scanning the memory of a process, and monitoring the encryption and decryption process enough times. With under ten iterations, RSA keys can be successfully extracted.

Both Klein (2006) and Kaplan and Geiger (2007) focus on finding RSA keys using their signature in the volatile memory of a computer. This is consistent with Halderman et al (2009), who used RSA structure to locate the keys in memory. However, they also show that additional encryption material can be found. They confirm that the contents of the DRAM of a computer can be retrieved even after the computer has been turned off for a certain amount of time. Furthermore, they propose a method to recover encryption keys from that content based on calculating the Hamming distance among certain number of contiguous bytes and if the distance is close enough, calculate the key schedule from those bytes, compare it with adjacent bytes and if they match, a key is found. They are successful in applying this technique to find AES and DES keys. A similar method is used by Maartmann-Moe et al (2009) to create the tool Interrogate. In their work, they can find AES, Twofish and Serpent encryption keys from the volatile memory. However, to identify Twofish keys in memory they need the use of Shannon entropy in addition to the key schedule calculation.

Something important to note is that much of the literature around forensics techniques to retrieve keys from memory focuses on extracting RSA keys which might not be useful when applied to ransomware since often, the malware is shipped with a public key that is used for encryption while its private counterpart is never present in the infected system. On the other hand, the techniques used by Halderman et al (2009) on AES have been shown to be effective in retrieving the data from encrypted files by ransomware (Davies et al., 2020). Therefore, it is useful to study the overlapping elements of the extraction of symmetric keys for both, digital forensics, and ransomware mitigation. Also, while much of the mitigation's techniques discussed in section 2.3.2 focuses on monitoring the cryptographic API (Bajpai, 2020; Continella et al., 2016; Kolodenker et al., 2017), the approaches discussed in this section provide an alternative method to manually retrieve the keys without interacting with the API.

Table 2 shows a summary of the studies that have developed a method of identifying and or extracting encryption keys from memory. It is shown that symmetric ciphers rely on key schedule or initial matrix state and that RSA suffers from a similar vulnerability with

a structure present in memory when the key is loaded. These vulnerabilities have been leveraged against ransomware, specifically with AES keys (Bajpai, 2020; Davies et al., 2020). By reviewing the types of ciphers used by ransomware, new methods of identifying keys in memory for those ciphers can be developed to be used as a mitigation tool. Although it is a static method, this approach could be used in cases where ransomware imports its own cryptographic library.

Table 2 - Memory Signature of Symmetric and Asymmetric Ciphers

Algorithm	Type	Signature in memory	Year of publication	Research
<i>Salsa20</i>	Symmetric stream cipher	Initial state matrix	2007	N/A
<i>Chacha20</i>	Symmetric stream cipher	Initial state matrix	2008	N/A
<i>AES</i>	Symmetric block cipher	Key schedule	2001	(Halderman et al., 2009; Maartmann-Moe et al., 2009)
<i>RSA</i>	Asymmetric block cipher	Key signature	1977	(Halderman et al., 2009; Hargreaves & Chivers, 2008; Nakano et al., 2014)
<i>DES</i>	Symmetric block cipher	Key schedule	1975	(Halderman et al., 2009)
<i>Serpent</i>	Symmetric block cipher	Key schedule	1998	(Maartmann-Moe et al., 2009)
<i>Twofish</i>	Symmetric block cipher	Key schedule and Shannon entropy	1998	(Maartmann-Moe et al., 2009)

## 2.5 Cryptographic Ransomware and Associated Research

Major recent surveys seem to identify the use of RSA and AES as the most popular ciphers among crypto ransomware (Beaman et al., 2021; Humayun et al., 2021; Mohammad, 2020; Moussaileb & le Boudier, 2021; Oz et al., 2021), with only technical analysis discussing other encryption mechanisms (Craciun et al., 2019; Yuste & Pastrana, 2021; Lee et al., 2019). Although crypto ransomware seems to be adopting more sophisticated ways of encryption. There seems to be a shift in the ciphers used by crypto ransomware. For instance, some versions of Petya use Salsa20 symmetric cipher to

encrypt the Master Boot Record (MBR) and Master File Table (MFT) of infected machines (Aidan et al., 2018; Fayi, 2018). Furthermore, GandCrab version 5 and Sodinokibi use Salsa20 to encrypt files (AMOSSYS Security Blog, n.d.; Lee et al., 2019). Darkside and Anatova have also been identified using Salsa20 cipher to encrypt files (*Darkside Ransomware* / Chuong Dong, n.d.; Poudyal, 2021), while others use Chacha which is a variant of Salsa20 (Yuste & Pastrana, 2021). This chapter compares some of the different encryption schemes that cryptographic ransomware uses and classifies it with the discussed categories in section 2.2. Moreover, this identification of new ciphers is contrasted with the current live memory forensics techniques as they could be potentially leveraged to mitigate this type of malware.

### 2.5.1 WannaCry

WannaCry is a crypto ransomware that exploits Windows Server Message Block (SMB) vulnerability and is also able to spread over LANs autonomously. During its original campaign, it targeted health, telecommunications, and other organisations (Rehman et al., 2018). WannaCry gained popularity with a second version that was released in 2017, affecting an estimate of more than 230,000 computers in 150 countries (Hassan, 2019). Later that year, Microsoft released a patch for this vulnerability but as Rehman et al note, cybercriminals can update the code to take advantage of a new discovered vulnerability (Rehman et al., 2018). Moreover, WannaCry derives its keys using the PRNG (pseudorandom number generator) function in the infected host and uses symmetric and asymmetric keys to achieve encryption, following the procedure explained in section 2.2.1 Figure 1, and exposing the private key (Bajpai, 2020; Genç, 2020).

There is only one study that claims to have extracted the private key that is used to encrypt all the symmetric encryption keys used for different files from memory; although, the code to do so is not available (Bajpai, 2020). On the other hand, there is literature showing that AES and RSA keys are present in memory dumps at specific times while it is being executed (Akbanov et al., 2019; Genç, 2020; Hsiao & Kao, 2018; Yuste & Pastrana, 2021).

### 2.5.2 GandCrab and Sodinokibi

GandCrab and Sodinokibi crypto ransomwares first appeared in 2018 and 2019 (Lemmou & Souidi, 2018; *Sodinokibi Ransomware Is Storming Organizations*, n.d.) and are attributed to the same group, namely REvil. (Ravikant & Alexander, 2021). Furthermore, they have a similar and peculiar feature, if they detect the language of a Commonwealth of Independent States (CIS) installed in the victim they avoid infection. Although, they differ on their cryptographic schemes GandCrab v1.0 and v2.2r use AES-256 to encrypt files. These keys are derived from TrueCrypt which comes with the malware (Lemmou & Souidi, 2018), avoiding the mechanisms that hook the cryptographic API discussed in

section 2.3.2. It then uses RSA public key, of a pair created using the crypto-API in the infected machine, to encrypt the symmetric keys. This behaviour has been identified only for versions v1.0 and v2.2r. The version 4 of the ransomware does not need initial connection with the server and uses Salsa20 for files and RSA-2048 public key embedded in the code to encrypt the symmetric keys of files (Usharani et al., 2021), it is unclear from the available sources whether versions in between and after use a similar strategy.

#### 2.5.2.1 Sodinokibi

Sodinokibi is an example of the threat that modern crypto ransomware poses using a complicated and robust cryptographic ecosystem (AMOSSYS Security Blog, n.d.). It uses Elliptic-Curve Diffie-Hellman (ECDH) key exchange to share secrets between the remote attacker and the victim. It also has a pair of session keys and two additional pairs that are created on the host machine, additionally, it ships with two public keys one in a configuration JSON file and the other embedded in the binary. Orchestrating these keys in a way that generates two additional AES keys that are ultimately involved in the creation of multiple Salsa20 keys, one for each file encrypted (AMOSSYS Security Blog, n.d.; Ravikant & Alexander, 2021). The objective of this complex and robust scheme is to create two parallel encryption mechanisms with two master keys that are independent of each other, so other criminals can use the malware, but the original developers can always intervene, retaining full control over their activities which can be considered an evolution from GandCrab (*Threat Spotlight: Sodinokibi Ransomware*, n.d.)

Theoretically, following the experiments of Bajpai (2020) and Davies et al (2020) attacking the key management of ransomware could be a potential countermeasure for the latest versions of GandCrab and Sodinokibi. This would involve retrieving the Salsa20 keys from memory while the data is being encrypted. A clear downside of this solution is the speed at which Salsa20 encrypts (Bernstein. D.J, n.d.), which would make the problem harder than finding AES or RSA keys.

#### 2.5.3 Petya

Petya is a family of crypto ransomware that first appeared in March 2016, although the major attack related to this malware began in June 2017 and affected 64 different countries (Aidan et al., 2018). Aidan et al (2018) and Cracium et al (2019) point out the different variants of this ransomware including, Red Petya, Green Petya, Petya, Petna, PetrWrap and NotPetya and more recent versions like BadRabbit which are particularly recognised by the encryption of the Master File Table (MFT) rendering the Master Boot Record (MBR) unusable by the system, causing the machine to crash and once it is rebooted the system loads into a Petya kernel, prompting the user for a ransom and denying access to the files.

### 2.5.3.1 NotPetya

Arguably, the most devastating strain of the Petya family was NotPetya in June 2017, becoming second global security issue in the world (Fayi, 2018). This version of Petya encrypts the file in the system as well as the MFT. However, there are existing mitigation techniques for this attack such as blocking SMB ports or using Decryptpetya.py tool (Fayi, 2018). Alternatively UShallNotPass denies the use of PRNG based on the legitimacy of the application, has also been shown to be effective (Genç, 2020; Genç et al., 2018).

NotPetya is a hybrid crypto ransomware that generates AES-128 keys derived from the Crypto-API in windows which is subsequently encrypted by a public RSA key embedded in the binary, after the keys are wiped (Adamov & Carlsson, 2017; Bajpai, 2020). However, some versions of the ransomware use Salsa20 to encrypt the MBR and MFT (Aidan et al., 2018). These versions of Petya are observed to store the Salsa20 key in the same space in memory and therefore are potentially recoverable (EternalPetya and the Lost Salsa20 Key | Malwarebytes Labs, n.d.). In this case Davies et al (2020) were successful in retrieving AES keys and decrypting files by using live memory forensics techniques.

### 2.5.4 Phobos

Phobos ransomware appeared in 2019 (*Phobos Ransomware, A Combo Of CrySiS & Dharma*, n.d.), it belongs to a previously known family with the name of Dharma. Malwarebytes (*A Deep Dive into Phobos Ransomware - Malwarebytes Labs / Malwarebytes Labs*, n.d.) notes that Phobos uses the cryptographic API in Windows to derive the AES symmetric keys to encrypt files. These findings are consistent with another more thorough work by Lee et al (2019) in which they note that there is also an embedded RSA-1024 key. The RSA key is used to encrypt the AES key once it has been used to encrypt files. The use of live memory forensics to recover AES keys and restore the encrypted files has also been successful against Phobos crypto ransomware (Davies et al., 2020).

### 2.5.5 CryptoWall

CryptoWall is a single key structure ransomware that first appeared in 2013. This ransomware uses Windows APIs to delete files which allows their recovery later (Hassan, 2019). More importantly, CryptoWall 3.0 and earlier versions use RSA-2048 (Oz et al., 2021; *The State of CryptoWall in 2018*, n.d.). This ransomware beacons the communications server, on hands of the malicious actors, to send information periodically and to retrieve a public RSA key to encrypt files. However, there are decryption tools available through the No More Ransom project (*CryptoWall Ransomware Threat Analysis / Secureworks*, n.d.).

This is an example of older crypto ransomware that does not use symmetric encryption for files and therefore, only the public key is present in memory. This means that the solutions based on monitoring the crypto-API or on live forensics techniques would not be useful to mitigate this crypto ransomware.

### 2.5.6 DarkSide

The FBI confirmed that DarkSide is behind the Colonial Pipeline attack in the United States on 2021 but it was first spotted in 2020 (*What We Know About Darkside Ransomware and the US Pipeline Attack*, n.d.). This ransomware is a hybrid type that uses Salsa20 to encrypt victim's files and RSA-1024 to encrypt the symmetric keys before all the volume shadow copies are deleted (Makrakis et al., 2021). Additionally, DarkSide checks if it is being run in a virtual environment to prevent being analysed dynamically (*Return of the Darkside: Analysis of a Large-Scale Data Theft Campaign*, n.d.)

Similar to Sodinokibi, this crypto ransomware is new and uses Salsa20 to encrypt files. Theoretically the use of live memory forensics could be used to retrieve the Salsa20 keys, although, the analysis would have to be carried out in a way that the virtual environment is not detected.

### 2.5.7 Taxonomy and Key Schemes

In chapter 2.2 of this work two main classification of crypto ransomware are discussed. These classifications consist of 8 categories by Zimba, Wang, et al (2021) and 6 categories by Bajpai et al (2018). In this section, such categories are used to compare the virulence of the strains of cryptographic ransomware analysed in chapter 2.4 with the aim to discover trends among them. Also, the previous use of live memory forensics against them is compared with the cryptographic scheme they use to discover any pattern and potential gaps that can be researched.

From Table 3 below, it can be observed that most modern ransomware are category 8 by Zimba, Wang et al (2021), meaning that they use a hybrid cryptosystem with local key generation and a public key embedded in the binary. On the other hand, Bajpai et al (2018) categorisation focuses on the flaws of the cryptographic scheme. For instance, if there are existing tools that can decrypt the files after the ransomware has encrypted files successfully. In this case, the passing of time may change how a specific strain is classified since new techniques and solutions are developed. From Table 3 it is shown that the most recent ransomwares are a category 6 (Bajpai et al., 2018) while older ransomware are lower categories. Furthermore, the use of memory forensics to mitigate ransomware has been successfully used for 3 of the listed cryptographic ransomwares (Bajpai, 2020; Davies et al., 2020). However, they have only recovered AES or RSA keys.

Table 3 - Ransomware Taxonomy and Live Forensics as Mitigation Technique

Strain and year	Categories BajPai(2020)   Zimba, Wang, et al (2019)		Cryptographic scheme	Key extracted from memory	Comments
<b>DarkSide - 2020</b>	Cat 6	Cat 6	Hybrid - RSA, Salsa20	N/A	N/A
<b>Phobos - 2019</b>	Cat 3	Cat 8	Hybrid - AES, RSA	AES keys (Davies et al., 2020)	N/A
<b>Sodinokibi -2019</b>	Cat 6	Cat 8	Hybrid -Salsa20, RSA, AES, ECDH	N/A	Decryptor tool is freely available
<b>GandCrab - 2018</b>	Cat 4	Cat 8	Hybrid -Salsa20, RSA, AES and RC4	N/A	Previous versions are Cat 3 / Cat 6 keys are not embedded in the binary.
<b>WannaCry - 2017</b>	Cat 3	Cat 8	Hybrid - AES, RSA	RSA and AES keys (Bajpai, 2020)	The procedure cannot be recreated from the literature.
<b>NotPetya - 2017</b>	Cat 6	Cat 8	Hybrid - RSA, AES, Salsa20	RSA and AES keys (Davies et al., 2020)	UShallNotPass and a decryption tool are successful against it. (Genç, 2020). Bajpai procedure cannot be recreated.
<b>CryptoWall 3.0 - 2013</b>	Cat 3	Cat 5	Asymmetric - RSA	N/A	Decryptor tool is available through No More Ransome project.

Moreover, recent cryptographic ransomware has been identified using Salsa20 to encrypt files; Darkside in 2020 (Makrakis et al., 2021), Sodinokibi in 2019 (Ravikant & Alexander, 2021), Ranzy Locker in 2020 (*A Detailed Walkthrough of Ranzy Locker Ransomware TTPs*, n.d.), Karma in 2021 (*Threat Thursday: Karma Ransomware*, n.d.), BlackMatter in 2021 (*BlackMatter Ransomware Analysis; The Dark Side Returns / McAfee Blog*, n.d.) and others such as NetWalker and Ragnar\_Locker (Yuste & Pastrana, 2021).

## 2.6 Conclusion

In conclusion, the methods of classifying ransomware provide a useful view of what types may be more damaging and therefore what aspects of the defence mechanisms should be evaluated to address this type of malware. For instance, hybrid crypto ransomware is more

difficult to mitigate and to recover from, which can also be placed in categories that consider if current tools for decrypting exist or if the implementation of their cryptographic schemes contain any flaws that can be exploited.

Furthermore, detection mechanisms are more prevalent in the literature when compared to reactive techniques and sometimes considered last resort options (Bajpai, 2020; Genç, 2020), it is safe to assume that stopping the malware as soon as it is detected results in less damage. On the other hand, reactive mechanisms are also important to explore since it is impossible to prevent every single case of infection and in these cases different solutions can be available to recover the data (Moussaileb & le Boudier, 2021). Also, new techniques are emerging such as using live forensics techniques to retrieve keys from memory with the subsequent retrieval of files (Bajpai, 2020; Davies et al., 2020; Yuste & Pastrana, 2021). Most reactive mechanisms efficiently focus on hooking the cryptographic API or monitoring the volatile memory of the system. However, there seems to be a gap in the literature around finding and extracting keys in memory for current ransomware when each file is encrypted with a unique key.

Digital forensics techniques have been shown to successfully mitigate crypto ransomware (Bajpai, 2020; Davies et al., 2020; Yuste & Pastrana, 2021). New methods may now be needed as ransomware is evolving, using new cryptography and more specific encryption keys (Oz et al., 2021). Currently there are documented methods to identify AES and RSA keys in memory, however, there seems to be a shift in the most recent crypto ransomware using the stream cipher Salsa20. With the motivation possibly due to its higher encryption speed (Bernstein, n.d.).



## 3. Experimental Design and Implementation

### 3.1 Introduction

The literature review found that live memory forensics techniques can be used to mitigate cryptographic ransomware. However, the use of these techniques to mitigate ransomware are not very popular. The use of memory forensics to identify the cryptographic keys used by ransomware does not depend on monitoring the crypto-APIs of the host as opposed to some of the reactive mechanisms described in section 2.3.2. Related work does not seem to have explored recent ransomware which tends to use unique keys to encrypt individual files. There seems to be a shift in the cryptographic schemes used by ransomware, with Salsa20 being the cipher used by many recent ransomware strains rather than AES.

This chapter develops a method and design based on the live forensic techniques discussed in the literature towards Salsa20 key extraction for ransomware mitigation. The first section describes the method with information relevant to the experiments and outlines the reasons for which they have been selected and what results are expected. The experiments aim to evaluate the forensic techniques discussed in section 2.4. This evaluation is done within the frame set out in the literature but with an upgraded design that considers the current algorithms used by ransomware discussed in section 2.5. These current techniques require new strategies to mitigate the ransomware, therefore a side channel attack is developed targeting the vulnerability of the cryptographic scheme of the malware. The design provides specific and detailed information about the environment that is used to execute the experiments, analysis, and possible setups. The initial experiments are carried out in a predictable manner and recreate the environment from the literature. The implementation of the experiments uses the same method on real ransomware samples, with unknown variables, producing data to identify and analyse new findings around the per file symmetric key encryption used in current ransomware.

### 3.2 Methods

Several studies have retrieved encryption keys from the volatile memory of target machines (Halderman et al., 2009; Hargreaves & Chivers, 2008; Kaplan & Geiger, 2007; Maartmann-Moe et al., 2009). Some have then focused on retrieving keys and to decrypt data encrypted by ransomware with this method. Digital forensics and ransomware research extracting keys from memory are overlapping fields. The method used by Davies et al (2020) bridges both fields of research and was successful in decryption of compromised ransomware encrypted files. This work was around AES ransomware key extraction from memory and one key per victim, so one AES key was used to encrypt all victim files. This work will focus on extracting encryption keys for the Salsa20 algorithm as this algorithm seems to be used by several of the recent ransomware, from the review of the literature and technical write ups. It also seems that many current ransomware

strains generate a key and nonce per encrypted files (*A Detailed Walkthrough of Ranzy Locker Ransomware TTPs*, n.d.; *BlackMatter Ransomware Analysis*; *The Dark Side Returns / McAfee Blog*, n.d.; Makrakis et al., 2021; Ravikant & Alexander, 2021). The retrieval of multiple keys to decrypt multiple files with a side channel attack seems to be a novel area and targets a gap in the literature as seen in section 2.4 where different mitigation and recovery techniques are reviewed. This would build on the previous related work on one key per victim, extending it to the current families of ransomware using per file symmetric encryption keys. The initial experiments are carried out to verify the implementation of the method that recreates the work of Davies et al (2020). The verification consists of identifying and retrieving AES keys from memory to then decrypt synthetic ransomware encrypted files. Similar methods are then developed for Salsa20 keys, and then aim to be applied to real ransomware sample memory towards mitigation.

Salsa20 was focused on in the work due to its increasing use by ransomware developers as highlighted in section 2.5 of the literature review. This could be due to the speed of the algorithm (Bernstein. D.J, n.d.). Salsa20 is a stream cipher, which encrypt data faster than block ciphers such as some modes of AES (Sharif & Mansoor, 2010). Salsa20 is also considered a lightweight cipher used for IoT devices with constraint resources (Reza et al., 2020). The speed of Salsa20 could be being adopted to help ransomware attacks avoid detection. Furthermore, there seem to be no tools publicly available that can retrieve Salsa20 keys from memory, and no related literature seems to cover methods to do so either. These reasons validate the work towards assessing the previously successful approach but for Salsa20 encryption keys. This may require the development of a tool that identifies Salsa20 keys in memory.

The method to identify and extract Salsa20 keys in memory consists of generating the keys in a machine and analysing the volatile memory find cryptographic material. As noted by Hargreaves and Chivers (2008), this method helps to establish the pattern of the objective material, in this case Salsa20. Firstly, by knowing the encryption key that needs to be searched and, secondly, by locating a pattern that can be used to find different keys created with the same implementation (Hargreaves & Chivers, 2008). This technique provides a solution that is independent of the platform and focuses on the algorithm itself.

A similar method can be applied to mitigate cryptographic ransomware. Tools to identify encryption keys in memory can be used to analyse the different memory dumps. However, in this instance, if cryptographic material is found, it is used to decrypt files encrypted by the ransomware.

An advantage of this method is that it allows analysis of the memory dumps with flexibility, with a range of different available tools, new customised ones, or even manually by examining the binary content. Also, it enables the capture of the memory dumps remotely with tools such as triage (<https://tria.ge/>) or for them to be moved to a more appropriate environment to be studied. Lastly, it removes potential ethical problems

when running malware as it has to be done in a rigorous and secure manner. In this way the malware can be run somewhere else or by a third party, where it is secure, and the memory dumps are analysed later.

It is important to note that different ransomware strains behave in different ways and that they may take a few seconds or minutes to begin encryption. This is particularly important since the encryption keys are present in memory only during or after the time of the encryption (Davies et al., 2020), if the ransomware does not delete or obfuscate those keys. In which case, the keys are in memory for the specific amount of time that is needed to encrypt the files. To verify that the encryption keys found are correct, the files encrypted by the ransomware are decrypted with those keys.

### 3.3 Experimental Design

Several experiments were designed, including initial experiments to replicate the previous work around capturing AES keys from memory, to explore similar methods for identifying and extracting Salsa20 keys. Further experiments were then carried out to apply the Salsa20 extraction methods to real running ransomware. Two different environments were designed, with the initial experiments using synthetic ransomware, and the real ransomware experiments requiring an air-gapped machine for ethical and safety reasons.

The designs of the initial experiments follow the method from the literature, similar to that of Davies et al (2020), and the real ransomware experiments build on this for ransomware using multiple encryption keys. Each experiment is designed to be carried out independently and in a controlled and reproducible manner.

In the initial experiments a key is generated and used to encrypt a file. This known key is then searched for in the volatile memory and used to decrypt the same file. If the method is successful then it is used in the second set of experiments, this time with unknown keys generated by real ransomware.

#### 3.3.1 Initial AES Key Extraction Experiment

The initial experiment was around replication of previous work (Davies et al., 2020), and creation of an environment, method, and tools to do this. The experiment will replicate ransomware behaviour by encrypting a file using a known AES key and IV pair, and then extracting memory and identifying the key, and finally validating the key by decryption of the encrypted file.

The experiment is divided in 3 main stages that are modular. The stages are as follows: First, a key and a file are created in a Virtual Machine (VM) and the key is used to encrypt the file. Note that a pair key and IV/nonce might be used depending on the encryption

type – AES or Salsa20. If an IV is used for encryption, it is appended to the beginning of the file which is a behaviour adopted by ransomware (Lemmou & Souidi, 2018). During the encryption, the VM is paused, a snapshot is taken. Secondly, the virtual memory extracted and analysed to cryptographic material; this is done with tools available from the literature. The third stage consist of decrypting the file that was encrypted in the first stage. This modularity provided by the stage approach should allow the experiments to be adapted to extract different encryption keys and expanded if needed towards multiple keys.

### 3.3.2 Initial Environment Design

The initial experiments aim to replicate the proposed methods by Davies et al (2020) for ransomware key capture from memory. Therefore, a Virtual Machine (VM) is used, the encryption of files is carried within it, and the memory can be acquired from the VM. Virtual Machines allow access to the virtual memory via the associated memory files. Furthermore, they have a snapshot option which can be used to revert the machine back to a previous state, which means the experiments can be done many times. This option can be used to perform many rounds of experiments with the same setup in short amount of time.

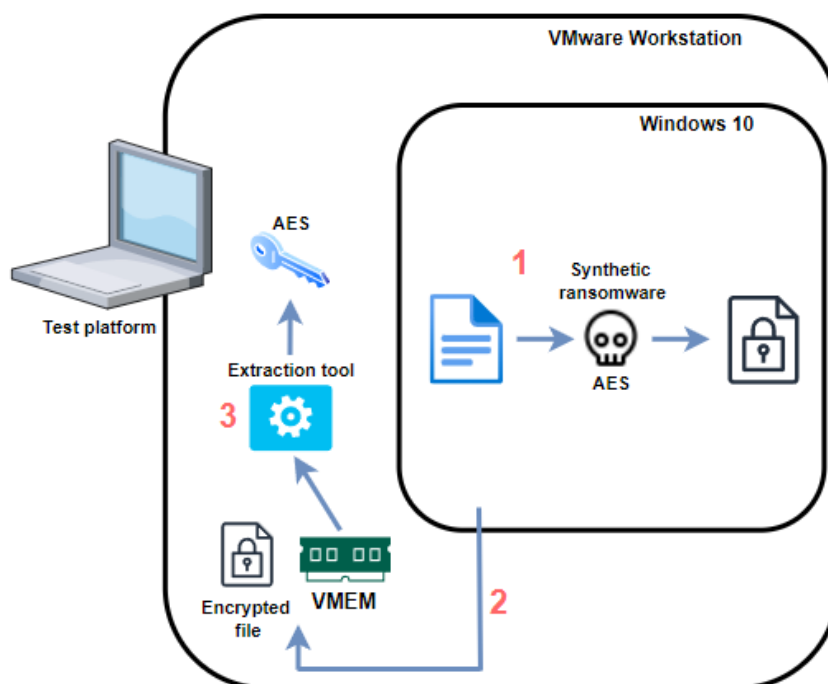


Figure 2 - Initial Environment

For the initial experiments a benign synthetic ransomware program is used to generate a key and encrypt the file, as shown in Figure two. Having control over the synthetic ransomware code that generates the keys enables the capability of encrypting with

different ciphers, and that the keys are ‘known’ so they can be searched for in the memory easily. Using synthetic ransomware code allows for the program to be paused during encryption, which makes finding keys in memory easier as they are present while they are being used by the program (Halderman et al., 2009). Finally, the extraction tool used is one those specified by Davies et al (2020). Another customised program is used to decrypt the encrypted file.

### 3.3.3 Technical Details of Initial Experiments

Table 4 shows the software selected to setup the initial environment. This information aims to help the reproducibility of the experiments and the understanding of the performance of the tools. On the left is the software selected and on the right is the version and other information that may be useful to reproduce the environment.

*Table 4 – Technical Details of Initial Experiments*

Software	Version and additional information
Hypervisor type 2	Vmware 15 Pro – 15.5.6 build-16341506
Python3	Version 3.9, Modules imported: Pycryptodome, time and os
Findaes	<a href="https://github.com/mmozeiko/aes-finder">https://github.com/mmozeiko/aes-finder</a>
Operating System	Windows 10 Education build 19041.1415
Virtual Machine	1 processor with 1 cores (i7-8700K), 1024 MB of RAM and 30 GB of dis space.
Hex Editor	HxD version 2.5.0.0
PowerShell	Version 5.1.19041.1320

### 3.3.4 Initial Experiment Stage 1 – Synthetic Ransomware using AES

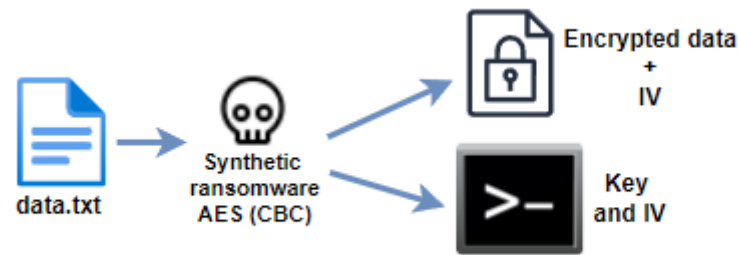


Figure 3 - Overview Stage 1 of Initial Experiment

In stage one the file was created, and a benign encryption program was used to generate an AES key and IV to encrypt the file inside the virtual machine in CBC (Cipher Block Chaining) mode. The program encrypts a file in a folder specified by the user. Furthermore, a time of 30 seconds was entered to keep the process alive in memory, ensuring that the key was loaded when the virtual machine was paused. Figure four shows the Key and IV generated by the program. However, the IV is also written to the

```

C:\Users\airbus a380\Desktop\ransomware>python encrypt_AES.py
Enter file name to encrypt: data.txt
Enter in seconds the time to pause the process: 30
Key: 91f5519ea957cba7b92eb58c7dec5806939e95647ccaf4b878b485195d862ac6
IV: 88e0d005d2bc3e2d7e35f42ffc956367
  
```

Figure 4 - Encrypt\_AES Output

beginning of the file which is behaviour similar to that seen in cryptographic ransomware (Ravikant & Alexander, 2021). Lastly, once the 30 seconds had elapsed the program wrote the IV and the encrypted content to a file on the directory. Figure five shows the original file opened with a Hex Editor with name “data.txt” and the after encryption with name “data\_enc.txt”

data.txt																
Offset(d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
00000000	4C	6F	72	65	6D	20	49	70	73	75	6D	20	69	73	20	73
00000016	69	6D	70	6C	79	20	64	75	6D	6D	79	20	74	65	78	74
00000032	20	6F	66	20	74	68	65	20	70	72	69	6E	74	69	6E	67
Lorem Ipsum is s imply dummy text of the printing																

data_enc.txt																
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	88	E0	D0	05	D2	BC	3E	2D	7E	35	F4	2F	FC	95	63	67
00000010	A6	B7	DE	2E	3F	E2	27	88	27	8C	7C	21	4C	7E	F7	38
00000020	55	2E	62	84	77	6A	43	6E	FC	43	87	52	A6	1B	3A	E5
^âD.Ô4>~5ô/ü•cg  ·P. ?â' ^'E !L~÷8 U.b„wjCnûC+R!.:â																

Figure 5 - File Before and After Encryption

### 3.3.5 Initial Experiment Stage 2 – Extract Memory and AES Keys



Figure 6 - Overview Stage 2 of Initial Experiment

In the second stage the VM was paused, and a snapshot taken so a “.vmem” was created. The extraction tool was Findaes since it was one of the tools used by Davies et al (2020). Findaes was used on the memory dump taken from the virtual machine *WindowsBeningRansomware-Snapshot.vmem*. The tool found additional results which could be other keys used by the operating system. Other keys and the known key generated in stage one, underlined in red, were found as shown in Figure seven. To find this key through the output a search pattern with the known key was done in the console.

```
Found AES-256 key schedule at offset 0x2c6bfd8:
f7 14 b8 a5 b5 e0 a1 d4 aa 50 cd 61 8c d5 7a 63 7a ee a9 26 01 65 8f 93 eb 92 4b a1 26 c4 79 f8
Found AES-256 key schedule at offset 0x3399e720:
91 f5 51 9e a9 57 cb a7 b9 2e b5 8c 7d ec 58 06 93 9e 95 64 7c ca f4 b8 78 b4 85 19 5d 86 2a c6
Found AES-256 key schedule at offset 0x37270358:
60 58 99 10 6e 2c df e4 d0 6b ac 55 46 1e 1b 6a a0 f9 91 8b 80 cb 8a 57 c6 f9 76 b1 79 cf d9 08
Found AES-256 key schedule at offset 0x372705e8:
be 4d a9 27 f4 c5 8d d2 e9 ff 6d 81 06 b8 be df 10 24 df 41 c9 05 eb 83 32 7f e7 dc f0 ea 22 24
```

Figure 7 - Findaes Output

### 3.3.6 Initial Experiment Stage 3 – Validate the Extracted AES Keys

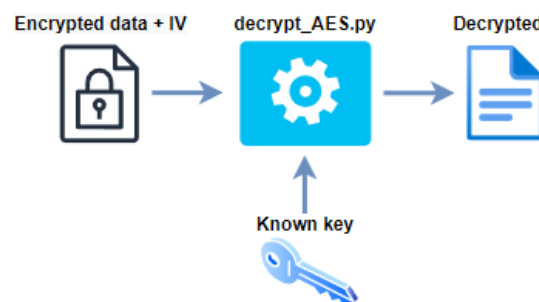


Figure 8 - Overview Stage 3 of Initial Experiment

In the last stage the encrypted file, named *data\_enc.txt*, was extracted from the virtual machine. Then a Python script, *decrypt\_AES.py*, was used with the recovered encryption key to decrypt the data. The decryption had the same AES mode to encrypt the data, in this case CBC. Note that the IV was added to the beginning of the file when it was encrypted, therefore the scripts read the 16 first bytes and used them as IV as shown in Figure nine.

```
with open(folder_path + "\\\" + file, 'rb') as file:
    IV = file.read(16)
    chunk = file.read()
    decrypted = decrypt(chunk, key, IV)
```

Figure 9 - Decrypt\_AES Code

Figure 10 shows the file being decrypted by the script and the decrypted data in hexadecimal compared to the original file named *data.txt*.

```
Enter file to decrypt: data_enc.txt
Enter key in hex: 91f5510ea037c8a7b92ab58c7dec5806d39e95647ccaf4b078b483193d862acc8
Decrypted text: b"Lorem Ipsum is simply dummy text of the printing and typesetting
Decrypted in hex: 4c6f72656d20497073756d2069732073696d706c792064756d6d7920746578742
```

data.txt	Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
	00000000	4C	6F	72	65	6D	20	49	70	73	75	6D	20	69	73	20	73	Lorem Ipsum is s
	00000010	69	6D	70	6C	79	20	64	75	6D	6D	79	20	74	65	78	74	imply dummy text
	00000020	20	6F	66	20	74	68	65	20	70	72	69	6E	74	69	6E	67	of the printing

Figure 10 - Decrypted Data and Original File

The initial experiments around AES were successful in recreating the method of Davies et al (2020), and although the process of encryption was not performed by real ransomware, it shows successful memory capture, and key extraction in the same way. These experiments demonstrate that the proposed ransomware experimental environment and method can achieve the expected results, in this case, encrypting a file, extracting the encryption key directly from memory, and reading the IV and decrypting that file with the recovered key. This process is what cryptographic ransomware does no matter how many keys or which algorithms are used. The initial experiments also should be able to be extended to possibly identify Salsa20 in memory and further to run real samples of ransomware.

### 3.3.7 Initial Experiments in Identifying Salsa20 Keys in Memory

The modularity of the above method means that each stage can be adapted to suit a specific need. For instance, a method to identify and extract Salsa20 key and nonce pairs can now be attempted before it can hopefully be used against a real sample of ransomware. This method is developed and tested without using real malware, in a similar way to the initial AES experiment, and then if successful tested against ransomware. The first step was to change the synthetic ransomware encryption to create Salsa20 keys and use those to encrypt files. Then to search for the Salsa20 keys in memory while the process was running. It is important to note that there seem to be no publicly available tools or published methods that can do this. However, with Hargreaves and Chivers (2008) logic, the pattern to find keys can be established from the manual analysis of the volatile memory. Therefore, a synthetic ransomware program was created this time to generate Salsa20 key and nonce pairs and encrypt files. This program is shown in Figure 11.



```
def salsa_encrypt(data):
    key = get_random_bytes(32)
    cipher = Salsa20.new(key=key)
    nonce = cipher.nonce
    encrypted = cipher.encrypt(data)
    print("Key: ", key.hex())
    print("Nonce: ", nonce.hex())
    return encrypted
```

Figure 11 – Generated Salsa20 Keys

The output of the script after encryption is as shown in Figure 12 below. This synthetic ransomware prompts the user for the amount of time to maintain the process in memory and prints out the key and nonce, as shown in Figure 12. Then, it reads the contents of the files and encrypts them but waits until the selected time has elapsed to write the encrypted data into a new file. This is done to attempt to analyse any difference in keys being loaded in memory for various different time periods the program is running.

```
C:\Users\airbus_a380\Desktop\ransomware>python encrypt_Salsa20.py
Enter file name to encrypt: data.txt
Enter in seconds the time to pause the process: 1
Key: af6f1443897fcf59452cf7e396c1938bfb76568b06fcad3df9426bbd1986e8ed
Nonce: a3b9f234dc08cb42

C:\Users\airbus_a380\Desktop\ransomware>python encrypt_Salsa20.py
Enter file name to encrypt: data1.txt
Enter in seconds the time to pause the process: 30
Key: 8e587e3411203d3a8be25e6b742fdf5a5f09c818e577dd9c05eee476558b8fd5
Nonce: e56655d7bd42f0b4

C:\Users\airbus_a380\Desktop\ransomware>python encrypt_Salsa20.py
Enter file name to encrypt: data2.txt
Enter in seconds the time to pause the process: 60
Key: ce38a08da222955a891d7d4a8a0572a7947372ec722c3a50ce1c88b816c642fb
Nonce: 4b5449c150a0343a
```

Figure 12 – Salsa20 Encryption Script

At first, captures of the memory were taken as the program terminated, but unfortunately the keys could not be found when these snapshots were examined manually. The entire Salsa20 key values did not seem to be in memory at this point.

Since the first experiment did not provide successful results through simply searching memory dumps for the keys, another experiment was carried out in which the snapshot was taken while the process was running, in a similar way to the AES. Figure 13 shows the key and nonce that were loaded in memory. After, the file `.vmem` was opened with a Hex Editor to look for the key and nonce.

```
C:\Users\airbus_a380\Desktop\ransomware>python encrypt_Salsa20.py
Enter file name to encrypt: data.txt
Enter in seconds the time to pause the process: 60
Key: b294bb081cf5b04d82f08286578071a82f6af41b56451fa1177513ec40d9bd38
Nonce: 5eecd8d240c0544
```

Figure 13 - Salsa20 being using for encryption

This time the entire key was found as shown in Figure 14. Although, it did seem to have any distinguishable pattern such as the key schedule that Halderman et al (2009) define being used to identify AES keys in memory, and that the Findaes tool uses.

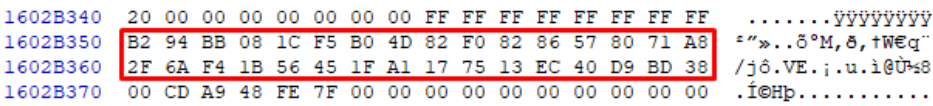


Figure 14 - Key Identified in Memory

Next, an attempt to use Shannon entropy, as proposed by Shamir and Van Someren (1999) to find Salsa20 keys produced an excessive number of false positives. Also, as the size of the memory dump increases so seemed to be the number of false positives. This method is comparable to brute forcing the memory for all possible key length values with high entropy, and therefore was considered unviable.

It is important to note that ransomware generates high amounts of entropy across the volatile memory due to the amount of file encryption being generated, so a general high entropy search like the above would probably not be useful (Halderman et al., 2009).

3.3.7.1 Identifying Salsa20 Initial State Matrix

Sections of the key values were searched for to attempt to locate all occurrences of the key in memory, in the memory dump and after further examination portions of the key were actually found in other places in the memory as well as the entire keys. The structure observed in memory seemed to contain a pattern with the strings “expa” and “nd 3”as shown in Figure 15. After this experimentation was performed several times, it was found

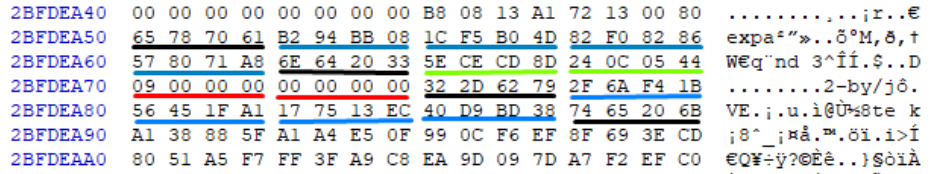


Figure 15 – Initial State Matrix in Memory

that this structure seems constant. After some more research into the Salsa20 algorithm it was found that this represents the Initial State Matrix of the key preceding its expansion function which is used to encrypt data (Bernstein, n.d.; Reza et al., 2020). Figure 16 shows a diagram of this Initial State which matched what was found in the memory dumps while Salsa20 is encrypting.

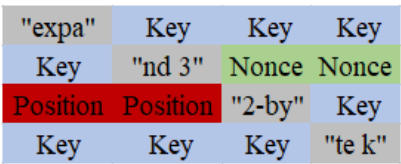


Figure 16 – Initial State Matrix Diagram

The matrix contains the encryption key, nonce, position of encryption and four constant strings, “**expa**”, “**nd 3**”, “**2-by**” and “**te k**”, which combined form “**expand 32 byte k**” as seen in Figure 16. This pattern is part of the specification of the algorithm and seemed to be something which could be used to find Sala20 keys in memory (Bernstein, n.d.). Interestingly the Salsa20 nonce is also stored in the structure.

### 3.3.7.2 Salsa20 Key Extraction Tool

After finding this possible signature for the Salsa20 Initial State Matrix, a tool to attempt to identify and extract Salsa20 key and nonce pairs was created. The tool initially searched for the Salsa20 Key State Matrix matching 4 character constant in the memory dump as shown in figure 17. The script accepted binary files as input which can be in different formats such as *.dmp*, *.vmem*, *.core*, etc. It then reads the file and converts bytes to hexadecimal values. After, the script searches for the string “**expa**”, which in hexadecimal is “**65787061**”. If there is a string that matches it, 128 hexadecimal characters starting from the beginning of the match are stored in a variable. These characters are equivalent to 64 bytes which make up the expansion function or initialisation matrix of the Salsa20 algorithm (Bernstein, n.d.). Figure 17 shows the Regular Expression used to match these bytes.

```
dump = f.read(chunk_sz)
chunk = dump.hex()
expand = re.findall(r'65787061.{128}', chunk)
```

Figure 17 – Pattern match the Salsa20 Initial State matrix

Once this was tested, more checks were added on the matched string for the rest of the key and nonce containing pattern. In this case the string “**nd 3**” which is “**6e642033**” in hexadecimal and word “**te k**” equivalent to “**7465206b**” must be in a specific position relevant to the rest of the structure defined as the Initial State Matrix above. These checks are shown in the code along with the key and nonce extraction in Figure 18.

```
for key in expand:
    if key[40:48] == '6e642033' and key[120:128] == '7465206b':
        filtered_keys.update({key[8:40] + key[88:120]: key[48:64]})
        # If second and fourth words exist in their location add to dictionary
        # key[8:40] + key[88:120] -> Key key[48:64] -> Nonce
for key, nonce in filtered_keys.items():
    if key not in storing_keys.values():
        storing_keys.update({key: nonce})
```

Figure 18 - Matrix Check

Once the script was used against the extracted memory dump, the key and nonce pair was found and extracted. Figure 19 shows the key underlined in red along its nonce below. As the memory files are large, reading the memory in large chunks was needed. To avoid splitting an Initial State Matrix structure in two when reading portions of memory, the script reads ahead and adds the 500 hexadecimal characters from the end of a chunk and adds them to the beginning 500 of the next. This new string of bytes is analysed in the

```
[+] Enter the name of the memory dump and its extension: WindowsDefender-Ransomware-Snapshot.vmem
[+] Finding Salsa20 keys loaded in memory...

[*] 32 byte Salsa20 Key - b294bb081cf5b04d82f08286578071a82f6af41b56451fa1177513ec40d9bd38
[*] Nonce - 5eecd8d240c0544
[+] Total number of keys found: 1
[+] That's all we could find!
```

Figure 19 – Extracted Key and Nonce

same way as the original, reducing the number of keys that the script might miss. However, this can increase the number of duplicated keys, so any duplicate keys are ignored.

### 3.3.7.3 Initial Experiment Stage 1 – Synthetic Ransomware using Salsa20

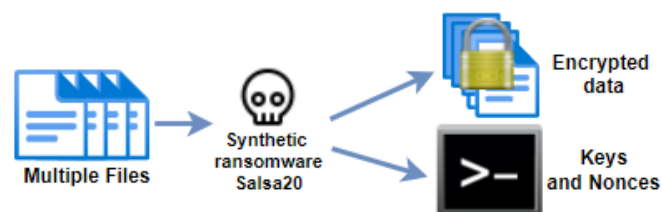


Figure 20 - Overview Stage 1 of Initial Experiment with Salsa20

Using the same Initial Experiment process used for the AES key extraction and validation, the new Salsa20 extraction method was evaluated. The synthetic ransomware previously used to encrypt files with Salsa20 was adapted to read all the files in the directory where it was executed. Then it encrypted the files and waited for a certain specified amount of time before writing the encrypted data to a file in the directory and terminating the process. To avoid replacing the files when written to the directory the program adds a counter number to the name of the file written. Additionally, it also outputted the Key and Nonce pairs to the console. Figure 21 shows this program generating four Salsa20 key and nonce pair, encrypting four files and terminating the process after 30 seconds.

```
C:\Users\airbus a380\Desktop\ransomware>encrypt_Salsa20.py
Enter in seconds the time to pause the process: 30

Encrypting: 02dataSalsa20.txt
Key: 6c2bbc2a19a6c33c34d37c3eff35a42c4d0432f2d82f0482b62bed2c7fed91dc
Nonce: 2460e49f3e01ac6f
Encrypting: 03dataSalsa20.txt
Key: f9d82fca88158f9afd8cae87ee843561bc7699c1867258e5a96cea821f65d897
Nonce: 81d4e1c4a56bd43a
Encrypting: 04dataSalsa20.txt
Key: 70b0b7f89696305ebfaacfb7e6df301600a99a6e5da0e84271565021bb434e9c
Nonce: 9a133a70728fce8a
Encrypting: 01dataSalsa20.txt
Key: fe77f966cb8516ea7baeb575154a56d0bb29926ba77f623092569b7099b32094
Nonce: 3d44157e37209d28
```

Figure 21 – Encrypting Multiple Files with Salsa20 Synthetic Ransomware

Finally, Figure 22 shows the header of one of the original files “02dataSalsa20.txt” and the encrypted version “2data\_enc\_Salsa20.txt”.

02dataSalsa20.txt	
Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000	4C 6F 72 65 6D 20 49 70 73 75 6D 20 69 73 20 73 Lorem Ipsum is s
00000010	69 6D 70 6C 79 20 64 75 6D 6D 79 20 74 65 78 74 imply dummy text
00000020	20 6F 66 20 74 68 65 20 70 72 69 6E 74 69 6E 67 of the printing
2data_enc_Salsa20.txt	
Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000	0D B6 A3 93 4C D1 7A D3 41 16 95 5F AC A0 FD 35 .qE"LÑzÓA.*_~ý5
00000010	DE 0D 19 CC C8 93 6B 33 51 76 2B 3E 86 F1 C5 13 P..iÈ"k3Qv+>+ñÄ.
00000020	0F D0 C3 5C D5 D6 DA 89 78 C8 D4 02 19 27 E9 88 .DÄ\ÖÖÜwxËÖ..'é^

Figure 23 – Before and After Encryption with Salsa20 Algorithm

### 3.3.7.4 Initial Experiment Stage 2 – Extract Memory and Salsa20 Keys



Figure 22 – Overview Stage 2 of Initial Experiment with Salsa20

This stage is similar to the stage 2 of the Initial Experiments with AES. When the synthetic ransomware was paused during the 30 seconds the Virtual Machine was also paused and a snapshot was taken so the `.vmem` file could be extracted. After, the extraction tool previously developed was used against the memory dump “`WindowsBeningRansomware-Snapshot13.vmem`”. Figure 24 shows the output of the extraction tool and that the key and nonce pair generated by the program in stage one was found and extracted successfully.

```
[+] Enter the name of the memory dump and its extension: WindowsBeningRansomware-Snapshot13.vmem
[+] Finding Salsa20 keys loaded in memory...

[*] 32 byte Salsa20 Key - 70b0b7f89696305ebfaacfb7e6df301600a99a6e5da0e84271565021bb434e9c
[*] Nonce - 9a133a70728fce8a
[*] 32 byte Salsa20 Key - 6c2bbc2a19a6c33c34d37c3eff35a42c4d0432f2d82f0482b62bed2c7fed91dc
[*] Nonce - 2460e49f3e01ac6f
[*] 32 byte Salsa20 Key - fe77f966cb8516ea7baeb575154a56d0bb29926ba77f623092569b7099b32094
[*] Nonce - 3d44157e37209d28
[*] 32 byte Salsa20 Key - f9d82fca88158f9afd8cae87ee843561bc7699c1867258e5a96cea821f65d897
[*] Nonce - 81d4e1c4a56bd43a
[*] Total number of keys found: 4
[+] That's all we could find!
```

Figure 24 - Extracting Salsa20 and Nonce pair

### 3.3.7.5 Initial Experiment Stage 3 – Validate the Extracted Salsa20 Keys

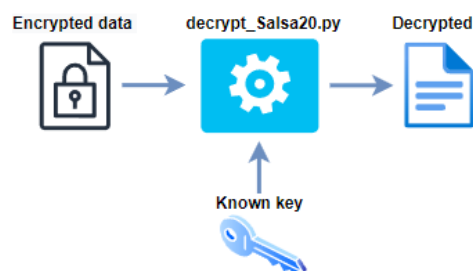


Figure 25 - Overview Stage 3 of Initial Experiment

To validate the extracted keys the files encrypted in stage one was decrypted with a Salsa20 script. The program prompted the user for a file name, key and nonce in hexadecimal and then outputted the data in plaintext and hexadecimal to the console, it also wrote the file to the directory. Figure 26 shows a comparison of one of the files “2data\_enc\_Salsa20.txt” after decryption data and the original file “02dataSalsa20.txt” encrypted in stage 1.

```

Enter file to decrypt: 2data_enc_Salsa20.txt
Enter key in hex: 8c20bc2a10a8c11c3ad17c3eff35aa2cad0a12f2d02f0402bd2c7fed91dc
Enter nonce in hex: 2a00a40f3e01ac6f
Decrypted text: b"Lorem Ipsum is simply dummy text of the printing and typesetting
Decrypted in hex: 4c6f72656d20497073756d2069732073696d706c792064756d6d7920746578742
02dataSalsa20.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 4C 6F 72 65 6D 20 49 70 73 75 6D 20 69 73 20 73 Lorem Ipsum is s
00000010 69 6D 70 6C 79 20 64 75 6D 6D 79 20 74 65 78 74 imply dummy text
00000020 20 6F 66 20 74 68 65 20 70 72 69 6E 74 69 6E 67 of the printing

```

Figure 26 – Decrypted Data with Salsa20 Algorithm and Original File

Figure 27 shows the decrypted file after it was written to the directory and compared byte by byte against the original file. In this case the decrypted file was named “data\_recovered.txt” and the original file “02dataSalsa20.txt”. The diff command was issued against both files and it showed that both files are identical, demonstrating the recovery of the original data.

```

airbus@DESKTOP-B6EE9T3:~$ diff 02dataSalsa20.txt data_recovered.txt
airbus@DESKTOP-B6EE9T3:~$

```

Figure 27 - Diff Against Decrypted and Original Files

The initial experiment for Salsa20 key recovery was successful in identifying and extracting the key and nonce pairs from memory. The method and tool developed was shown to be successful in extracting keys from the memory of a running synthetic ransomware program. It is important to note that there does not seem to be any literature regarding this concept, and this seems to be a novel method and tool. The tool developed has a similar concept to those used to extract RSA or AES keys from memory with their key schedule matching on a pattern in memory (Halderman et al., 2009; Hargreaves & Chivers, 2008; Kaplan & Geiger, 2007; Maartmann-Moe et al., 2009).

The method was successful in extracting the Salsa20 key and nonce from the captured memory and also the recovered key and nonce pairs were shown to be able to decrypt the synthetic ransomware encrypted file. After used successfully to decrypt the synthetic ransomware encrypted file. After creating and successfully testing this new method with the synthetic ransomware, it was decided to attempt to use this with real ransomware samples to evaluate its usefulness towards real ransomware mitigation.



### 3.4 Real Ransomware Experiments Design

These experiments aim to use the new method of identifying Salsa20 keys in memory and leverage it against real cryptographic ransomware samples. Current ransomware which uses the Sala20 cipher was found to also generate multiple number of key and nonce pairs to encrypt files with each of those unique pairs, so this also had to be considered, and to create a side channel attack as explained in section 3.2 to recover the multiple key and nonce pairs.

A specific strain of ransomware was selected for the purpose of extracting multiple keys and executed in a safe experimental environment, which was created for this work and based on best practises. To evaluate the identification of Salsa20 keys in the memory of the running cryptographic ransomware, snapshots will be taken at time intervals while the ransomware is running and then the memory dumps will be analysed to identify Salsa20 key and nonce pairs. These intervals will then be adjusted in further experiments to attempt to discover when, how many and how long keys are in memory. If keys are recovered, the next step is to verify them by decrypting the files encrypted by the real ransomware. This may need further research into the specific strain, as cryptographic ransomware tends to use different methods of encryption and add additional metadata to encrypted files (Oz et al., 2021). Lastly, to create a timeline and identify how long or when encryption keys can be found in the volatile memory the experiments are repeated, with different time intervals at which the memory is captured. Also, a data set of files to be encrypted including number and types of files has to be considered. Timeline experiments will be based on numbers of keys identified at each time interval, and not focused on the validation of the recovery of the encrypted files.

#### 3.4.1 Real Ransomware Environment Design

The environment design aims to create a test setup to analyse malware in a secure manner. When analysing malware, the environment should be as similar to the real world as possible (Rossow et al., 2012), but has to be safe and not confine the malware running. The emulation of an operating system in a virtual machine solves this problem in most cases. Although, some malware may use detection techniques to evade analysis in a virtualised environment (Raffetseder et al., 2007) and some may even try to escape the virtual environment. Ideally, the sample of ransomware selected is studied in advance via the available literature, to find out if it is likely to have this capability. Additionally, the virtual machine should be hosted in an air-gapped system, if possible, to add another layer of protection. As noted by Sikorski and Honig (2012), an air-gapped machine lacks any external connections to which malware can spread. This is an extra safety feature of the design to prevent ransomware breaking out of the virtual environment unexpectedly and causing damage or spreading to other systems.

Other important characteristics when analysing malware is to have a clear purpose and definition of the problems to be solved or explored (Berkay Celik et al., 2018). The purpose and definition of the problems be consulted in the method in section 3.2 and in the objectives of this document. A correct set of specific parameters may be needed to produce the desired results, so it is important to consider them as part of the experimental design noted by Berkay Celik et al (2018). For instance, a sample of ransomware might need internet connection to function, or it might need a specific amount of computing resources or run for a certain amount of time. These experimental parameters and even the environment may need to be adjusted and several experiments run accordingly towards the sought after results. For instance, creating additional virtual machines that simulate internet connection with services such as DNS, HTTP, HTTPS and others (Sikorski & Honig, 2012). Lastly, both, Rossow et al (2012) and Berkay Celik et al (2018) agree that the validation of the experiments is especially important in producing empirical results. The validation in this work includes the retrieval of the encrypted data and matching against the original input file dataset.

The environment proposal for running experiments on real ransomware samples is similar to the previous environment but in addition the laptop running the VM will be disconnected from all networks to ensure the containment of the malware and be as safe and ethically responsible as possible. To achieve this, an air-gapped laptop is prepared with a fresh updated operating system, and a similar VM running on it. It is important that this system does not have sensitive information that can be lost. Additionally, the network interface card is disconnected from any network and disabled, and the laptop is switched to flight mode. A hypervisor type two is installed, and a virtual machine created. This virtual machine is prepared with a new operating system with the last updates, the test files, and the sample. After, a snapshot is taken reducing the chance of accidental damage by transporting the ransomware sample multiple times. This setup increases efficiency allowing multiple experiments to be carried out consecutively in short time since the state of the virtual machine can be reversed to the initial state.

The only connection that the physical computer has with the exterior is an USB stick. This connection is only used to move the ransomware sample and test files into the virtual machine inside the virtual machine and to extract the virtual memory and encrypted files for analysis. Figure 28 shows the layout of the experimental environmental for running real ransomware samples. It shows the sample is downloaded from Triage (<https://triage.ge/>) into the analysis platform, and then moved to the air gapped test platform via USB stick. The sample is downloaded in zip format and does not have an executable extension and is only unzipped on the test platform which avoids accidental execution. The ransomware sample is run in the VM on the test platform and the memory captures and encrypted files are moved back to the analysis platform via USB also. The analysis machine is then used to carry out the experiments to extract keys from the memory dumps and validate the encryption keys found. Lastly, the hypervisor type two



software is updated to prevent the ransomware from using a known vulnerability to break out of the virtual machine.

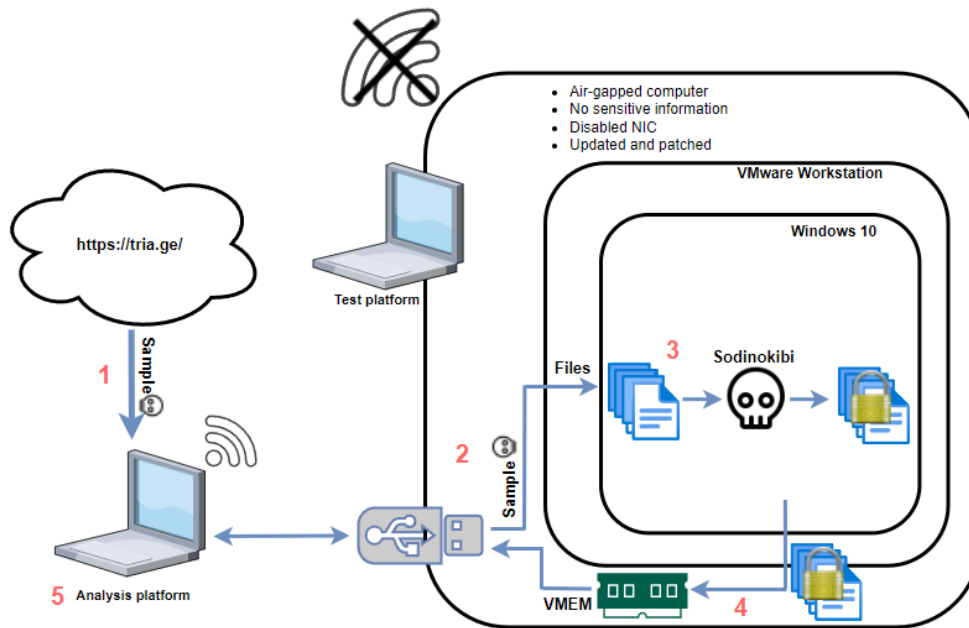


Figure 28 – Real Ransomware Environment

### 3.4.2 Salsa20 Ransomware Sample

Section 2.5 highlighted the ransomware behaviour and identified different strains of cryptographic ransomware. Sodinokibi was identified in the literature review as using Salasa20 and being deployed at the end of 2021. It appeared in 2019 and has been identified to use complex hybrid cryptographic scheme (Ravikant & Alexander, 2021). Sodinokibi uses a unique Salsa20 to encrypt each of the victim's files after infection (AMOSSYS Security Blog, n.d.; GandCrab V4.0 Analysis: New Shell, Same Old Menace, n.d.). However, this cryptographic ransomware uses a wide range of encryption algorithms, orchestrating them to avoid leakage of relevant information, and to ensure that the files can only be recovered with one of the two master keys which are never present on the victim machine. This would typically make the decryption of files impossible without the master keys. Additionally, the encryption of files is performed in a multi-threaded manner with the aim of making the encryption faster (Ravikant & Alexander, 2021). Figure 29 (AMOSSYS Security Blog, n.d.), shows the process that is used to derive the unique Salsa20 key and nonce used for each file.

From the literature and technical analysis write ups some details of the Sodinokibi encryption and metadata creation are defined. At first it seems to create a per file public and private key and a session key. It then uses the private key and the session public key with the Elliptic-Curve Diffie-Hellman (ECDH) algorithm to generate a file encryption key which is used to derive the Salsa20 key and nonce pair that used to encrypt the data content of each file. After encryption, it adds metadata at the end of each encrypted file

including the plaintext nonce and encrypted Salsa20 key, as well as some configuration parameters (AMOSSYS Security Blog, n.d.). While reversing the encryption of the Salsa20 keys is impossible, given that the private master key is never on the victim machine, a side channel attack should be able to be performed by capturing the Salsa20 key directly from the volatile memory while the malware encrypts the files, as shown in our earlier experiments.

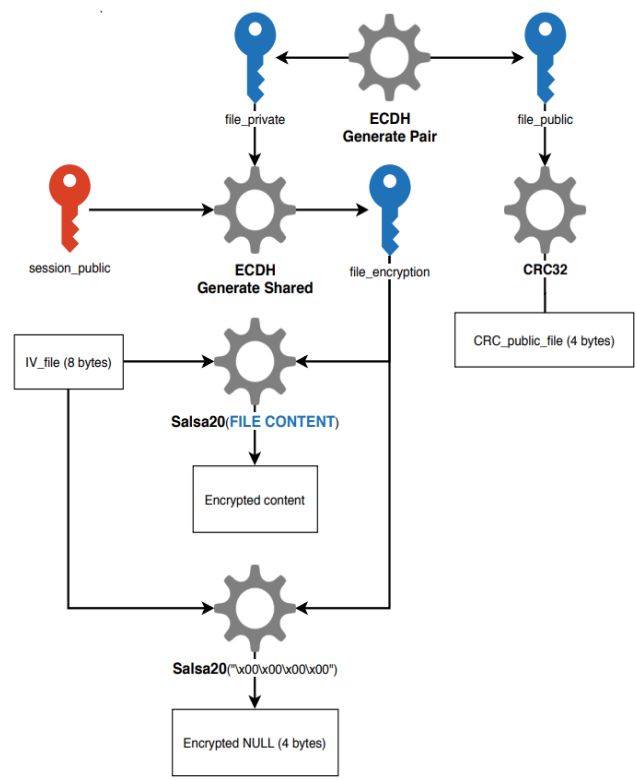


Figure 29 - Sodinokibi Encryption Process

3.4.3 Technical Specifications for Real Sample Experiments

The technical details for this and following experiments is specified in Table 5. The specific sample of the Sodinokibi ransomware is defined by its hash as previously discussed. The specific sample and other information around the VM and environment and tools configuration aim to help with the reproducibility of the experiments.

Table 5 - Technical Details of Experiments

Software	Version and additional information
Hypervisor type 2	VMware 15 Pro - 15.5.6 build-16341506

Python3	Version 3.9, Modules imported: PyCryptodome, Time and Os
findaes	<a href="https://github.com/mmozeiko/aes-finder">https://github.com/mmozeiko/aes-finder</a>
Operating System	Windows 10 Education build 19041.1415
Virtual Machine	1 processor with 1 cores (i7-8700K), 1024 MB of RAM and 30 GB of dis space.
Hex Editor	HxD version 2.5.0.0
Sodinokibi	Hash: 96dde0a25cc6ca81a6d3d5025a36827b598d94f0fca6ab0363bfc893706f2e87

### 3.4.4 Experiment 1 – Sodinokibi Ransomware Salsa20 Key Recovery

#### *Stage 1 – Sodinokibi File Encryption*



Figure 30 - Overview Stage 1 – Sodinokibi File Encryption

The method of extracting Salsa20 key and nonce pairs has been shown to work with a benign synthetic ransomware encryption program in a controlled environment. To evaluate the new Salsa20 key identification and extraction method with the real Sodinokibi ransomware sample, the same three stage experiment was carried out in the air gapped real sample experimental environment. Furthermore, a data set of files to be encrypted are created on the virtual machine. The files are from a dataset of mixed files designed to use in ransomware experimental work (Davies et al., 2021a). Initially only three files were used as the data set, to work through the main experimental steps of key

extraction and validation of decryption and find if these were possible, before moving onto a larger data set of files.

```
C:\Users\ExperimentExpand>cd Desktop
C:\Users\ExperimentExpand\Desktop>96dde0a25cc6ca81a6d3d5025a36827b598d94f0fca6ab0363bfc893706f2e87.exe -path files
C:\Users\ExperimentExpand\Desktop>_

----- Welcome NinaShoes. -----

[+] Whats Happen? [+]

Your files are encrypted, and currently unavailable. You can check it: all files on your system has extension g527ec5dv1.
By the way, everything is possible to recover (restore), but you need to follow our instructions. Otherwise, you cant return your data (NEVER).

[+] What guarantees? [+]

Its just a business. We absolutely do not care about you and your deals, except getting benefits. If we do not do our work and liabilities - nobody will not cooperate with us.
To check the ability of returning files, You should go to our website. There you can decrypt one file for free. That is our guarantee.
If you will not cooperate with our service - for us, its does not matter. But you will lose your time and data, cause just we have the private key. In practice - time is much

[+] How to get access on website? [+]

You have two ways:

1) [Recommended] Using a TOR browser!
a) Download and install TOR browser from this site: https://torproject.org/
b) Open our website: http://ap1ebzu47wgazapdqks6vrcv6zcnjppkxbxr6wketf56nf6aq2nmyoyd.onion/CC3656C04CE8F6F5

2) If TOR blocked in your country, try to use VPN! But you can use our secondary website. For this:
a) Open your any browser (Chrome, Firefox, Opera, IE, Edge)
b) Open our secondary website: http://decryptor.cc/CC3656C04CE8F6F5
```

Figure 31 - Sodinokibi Executed

Figure 31 above shows the real Sodinokibi sample running in the air gapped test environment. Figure 32 below shows the data set of three files before and after encryption.

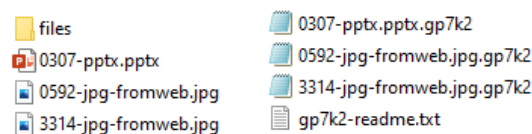


Figure 32 - Files Encrypted by Sodinokibi

## Stage 2 - Memory and Salsa20 Keys Extraction



Figure 33 - Overview Stage 2 – Salsa20 Key Extraction

Four memory dumps were taken at 10, 20, 30 and 40 seconds after the execution of the sample as it was observed after initial experiments that the full encryption had completed within 40 seconds. The script to find Salsa20 key and nonce pairs in the memory dumps was used with successful results for the dumps taken at 20, 30 and 40 seconds. The ransomware may do some reconnaissance of the system before beginning encryption (Nissim et al., 2019). This may explain if earlier memory dumps do not contain relevant cryptographic keys. The results for each of the memory dump analysis is outlined in

Figure 34. It is shown two key and nonce pairs were found at 20 seconds, then a new pair is additionally found at 30 seconds, and the last dump, at 40 seconds, only has the last pair found at 30 seconds, with perhaps the previous keys having been deleted from memory as described by some of the technical papers on this ransomware.

**At 20 seconds**

```
Key: 232371c7f5194c40f2c7696b4e48604439a8da6a32290044f4dd1b3a387e00f4
Nonce: 3d61027bb8265ce8

Key: edd8a9f8e446c41b312649df34f5237740db6b2a5923a5a44300389c9c3d3f54
Nonce: 094f60cabbcbfb4c9
```

**At 30 seconds**

```
Key: 232371c7f5194c40f2c7696b4e48604439a8da6a32290044f4dd1b3a387e00f4
Nonce: 3d61027bb8265ce8

Key: edd8a9f8e446c41b312649df34f5237740db6b2a5923a5a44300389c9c3d3f54
Nonce: 094f60cabbcbfb4c9

Key: 6d38d2f2d0318e53712156ad06f6813349be934a83b15f723f039a82fa3f2fb9
Nonce: 1c5cd305932039f5
```

**At 40 seconds**

```
Key: 6d38d2f2d0318e53712156ad06f6813349be934a83b15f723f039a82fa3f2fb9
Nonce: 1c5cd305932039f5
```

Figure 34 - Keys Extracted from Sodinokibi

Also pointed out by the literature, the appended data at the end of the files contains the nonce used to encrypt them, as well as some other metadata information related to the ransomware and encryption keys. (A Deep Dive into Phobos Ransomware - Malwarebytes Labs | Malwarebytes Labs, n.d.; AMOSSYS Security Blog, n.d.).

This piece of information was used to perform an iterative search over the encrypted files looking to match the recovered nonce's to the encrypted files. The search found the three files with matching the nonce appended as shown in Figure 35. This finding demonstrates a relationship between the nonce and key pair and these specific Sodinokibi encrypted files. So, the encryption keys used to encrypt those files have been identified and should now be able to be used to decrypt the specific files.

```
[+] Enter folder with files: files
[*] Nonce: 1c5cd305932039f5 in file: 3314-jpg-fromweb.jpg.gp7k2 found.
[*] Nonce: 094f60cabbcbfb4c9 in file: 0592-jpg-fromweb.jpg.gp7k2 found.
[*] Nonce: 3d61027bb8265ce8 in file: 0307-pptx.pptx.gp7k2 found.
```

Figure 35 – Salsa20 Nonce and File Related

### Stage 3 - Validating Extracted Keys by Decryption of Files

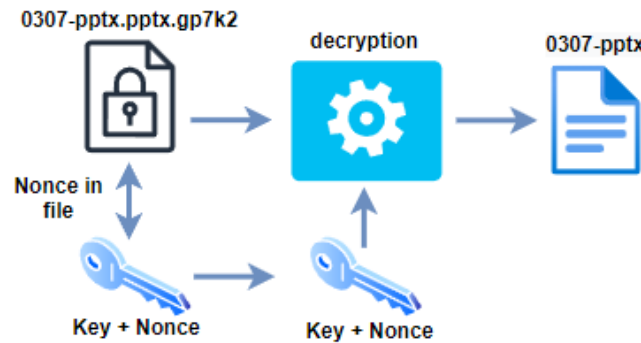


Figure 36 – Overview Stage 3 - Key Validation

This is stage 3 of our experiment and is aimed to validate the Salsa keys recovered from memory are the keys used for encryption. Similarly, to previous experiments this is done by attempting to decrypt the Sodinokibi encrypted files with the recovered keys and nonce pairs from the previous stage. After a comparison of the appended data structure with the details in a Sodinokibi technical paper (AMOSSYS Security Blog, n.d.), the files with the nonce appended was analysed using a hex editor to identify all of the metadata. This is shown in Figure 37 which highlights the nonce used for encryption for the file **0307-pptx.pptx.gp7k2** and can be located 16+8 bytes from the end of the metadata. From this

```

03151264 48 7E 6C 6A 36 73 E7 4B 39 17 F0 CD DD A9 10 C3 H~lj6sqK9.6iY@.Å
03151280 B7 2D 1C F1 6D 59 0E DC C1 82 E8 D4 12 0B 43 E9 -.fmY.ÜA,eÖ..Cé
03151296 0A BF C9 44 6F F6 52 BB 77 E7 33 C8 B3 F4 B7 BE .zEDoöRwqç3È³ö¼
03151312 83 AF 2C F8 52 53 56 6F 39 C1 26 E0 46 F2 3B 7F f~,øRSVo9Å4aFø;.
03151328 5E B3 06 53 04 E6 A6 5C 50 C5 DE 49 BA BD E1 7A ^'.S.æ!\PÄPI*ãáz
03151344 86 5E 4B 57 EF 91 F9 A0 1F 07 76 6B CA 66 8C 71 t^KWl'ù ..vkÈEeq
03151360 A9 0F D5 C5 29 1C 37 BC 57 48 0B F8 54 D2 C8 87 @.ÖÅ).74WH.øTÖE+
03151376 1C EA 30 14 A1 5C 56 2A 8E 03 93 E1 A9 65 D4 7D .è0.;\V×Z."á@eö)
03151392 FF E2 20 80 55 F9 6F 49 4F 36 B4 62 4A BC E7 84 yÄ EUúoIO6'bu4ç„
03151408 7D C7 BD 71 23 67 D0 52 48 B6 57 7B DA 57 6D AF }Ç*sq#gDRHqW{ÜWm~
03151424 0D 83 56 55 27 E8 13 31 AD 08 0A FE 22 37 7C 1B .fVU'e.l...p"7|.
03151440 14 7D 8E 82 89 23 3E 9D 86 E6 D6 B2 D5 EE 3E 1D .)Z,t#>.tæO*Öi>.
03151456 2B A9 A9 02 3A DD 93 6B 41 73 76 20 A8 45 78 84 +@@.:Y"KAsv `Ex„
03151472 0F 69 F6 7C C7 D0 3D F9 45 E9 CA E5 8C AD 41 63 .iö|ÇD=ùEéEÄE.Ac
03151488 73 D2 BC 62 BE 60 B8 0E 50 56 7A 8E D6 FB 1A 33 sÖ4b4".,FVzZÖü.3
03151504 F4 84 E6 2F 92 80 F5 8F 9B 7D 2D D3 69 F6 13 30 ö„æ/'eö.~)-Öiö.0
03151520 69 70 F5 33 31 FD BC A4 1D FF 00 47 45 29 A2 BD ipö3lY4+M.y.GE)ç%
03151536 3D F4 A9 B9 19 27 09 B2 BF 0C 89 0D 9D 1A 74 6F =@@'.'.ç.k...to
03151552 E6 1A 39 CD 3F 5F 28 FD 89 B3 30 3B 40 2D 53 8B æ.9i? (yñ³0;@-S<
03151568 F2 EE 57 CC E3 B8 79 98 BB 12 B4 0D 35 AD 1F 55 öiWlÄ.y"».'.5..U
03151584 2F 23 F7 E6 59 9F 4B C2 FA AE 61 10 03 3D 0E BB /#±æYKÄú@a...=»
03151600 70 BE F4 A5 10 9C B0 2E 13 31 49 1E 73 C4 90 5D p%öY.æ°...lI.sÄ.]
03151616 BB F5 1D 60 9A 8C FF 55 B2 67 C5 F8 9E E6 D1 99 »ö.'sæYU+gÄæzæN™
03151632 95 C8 31 E2 6E 8D 53 1D 10 7D 38 81 33 EE C8 95 •E1án.S...}8.3iE•
03151648 31 3F 33 49 3C 1B F9 2F 73 C3 25 09 F5 BB 38 6B l?3I<.é.sÅ%.ö»8k
03151664 0C FA C2 16 E6 DC 1C 0E 84 91 36 23 20 DA 0B 0F .úÄ.æÜ... '6# Ü..
03151680 DF 66 01 9B 5C DE 63 FE 82 26 C4 5B DB D2 DE 47 Bf.>\Ecp,6Ä{ÜÖBG
03151696 71 62 3B 18 2E 05 F8 0D 3D 61 02 7B B8 26 5C E8 qb;...ø.=a.{,6\è
03151712 2F 94 F2 93 00 00 00 00 00 00 00 00 27 D7 F3 80 /"ö".....>×öE[]

```

- Encrypted content
- Secret 1 - 88 Bytes
- Secret 2 - 88 Bytes
- File public - 32 Bytes
- Nonce - 8 Bytes
- CRC public file - 4 Bytes
- Encryption type - 4 Bytes
- Spsize - 4 Bytes
- Encrypted NULL - 4 Bytes

Figure 37 - Sodinokibi Appended Metadata Structure

figure it can be observed that the bytes of the appended data match in size and structure with those found in the files. For example, the sum of the bytes of all the appended data is equal to **232** bytes which equals the difference between the original size of the file **0307-pptx.pptx.gp7k2** before encryption **3,151,496** bytes, and after encryption **3,151,728**, as seen in Figure 38.

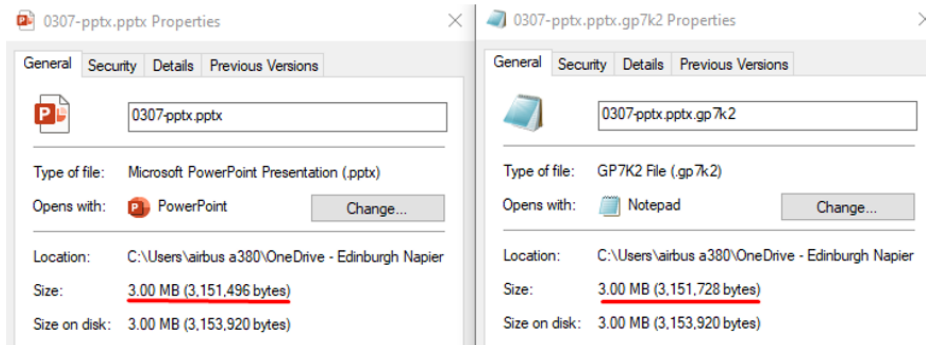


Figure 38 – File Size Before and After Encryption

To decrypt the files, it was first necessary to analyse the encrypted files for the metadata, to match the nonce and then decryption of the encrypted files could be attempted with the encryption key found with that specific nonce in memory. At this point, the key and nonce pairs and the files related to those pairs had been gathered and the decryption was attempted using the Salsa20 decryption script used in the Initial Experiments of Salsa20.

Recovering the encrypted data was not a straightforward task. Attempting to decrypt the files using the extracted key and nonce pairs by simply removing the metadata and decrypting the encrypted content, did not work. The encrypted file content was the expected size, being the same size in bytes as the original file, but decryption starting at the beginning of the file produced no readable plaintext. Similar to the problem of extracting Salsa20 keys from memory, there is no literature or technical documentation on specifically how to decrypt files encrypted by Sodinokibi samples, only on the metadata. The literature only stated the key and nonce were used to encrypt and could be used to decrypt. While there is a decryption tool available for this version of Sodinokibi, the author, Bitdefender, state that there is an ongoing investigation against this cryptographic ransomware involving other law enforcement groups and that they cannot make comments on it (*Bitdefender Offers Free Universal Decryptor for REvil/Sodinokibi Ransomware*, n.d.). Many detailed technical write ups on the encryption exist, and were reviewed, but none had any more details.

The logical process was to begin encryption from the beginning of the file, and then byte by byte to try and find some part of the input file. The file was opened and read byte by byte via the decryption program, using the key and nonce relative to that file. If there was a match with the header bytes of the specific input file, the file was written back with the relevant extension. However, this resulted in corrupted data when applied to Sodinokibi



encrypted files. The magic numbers/file header may have been removed as with some ransomware, so this did not work.

### Metadata Null Bytes Checksum Encryption Analysis

From the appended metadata structure shown in Figure 37 above, it was observed that the last 4 bytes of the appended data with label “**Encrypted NULL**” was used by the ransomware to check that the encryption is correct after is completed (AMOSSYS Security Blog, n.d.). Therefore, to verify the key, nonce pair found by the extraction tool further, the decryption of the last 4 bytes of the encrypted file should result in “00 00 00 00”. Figure 39 shows a script that has the null values of the file 0307-pptx.pptx.gp7k2.

```
plaintext = '27D7F380'
plaintext_bytes = bytes.fromhex(plaintext)
key_hex = '232371c7f5194c40f2c7696b4e48604439a8da6a32290044f4dd1b3a387e00f4'
secret = bytes.fromhex(key_hex)
nonce_hex = '3d61027bb8265ce8'
nonce = bytes.fromhex(nonce_hex)
cipher = Salsa20.new(key=secret, nonce=nonce)
decrypted = cipher.decrypt(plaintext_bytes)
print('Decrypting "27D7F380":', decrypted.hex())
```

*Figure 39 - Decrypting Null Bytes*

These values were stored as a variable with the name “*plaintext*”. The key and nonce pair recovered from the memory and related to the file were also used. The result showed the 4 bytes with zero values, this result verified that the same key and nonce pair were used to encrypt the file. This was also done for the three files for which the key and nonce were found and the same result were reached. Furthermore, the experiment seems to show that the Salsa20 implementation used in this script was the same used by this sample of Sodinokibi and that the key and nonce pair were used together to encrypt the file.

After this, another process that was tried consisted of removing the metadata, 232 bytes from the end of the file, before decrypting it. In a different attempt, the file was stripped byte by byte from the end and decryption tried after each step. This process also tried stripping the bytes from the beginning. None of these attempts found the magic numbers of the files that were encrypted using the key and nonce pair extracted from the memory dumps.



Salsa20 is a stream cipher and therefore it encrypts and decrypts one byte at a time (Bernstein, n.d.). Also, decryption and encryption are the same process for stream ciphers. The length of the data and the position in which the encryption or decryption begins must be the same, otherwise the returned bytes will not match the original data. Figure 40 (*Salsa20 Block Diagram. / Download Scientific Diagram, n.d.*) shows a summary of the Salsa20 function, although it consists of 20 rounds and not 4. The expansion function, or Initial State Matrix, expands the key with which the data will be encrypted after the

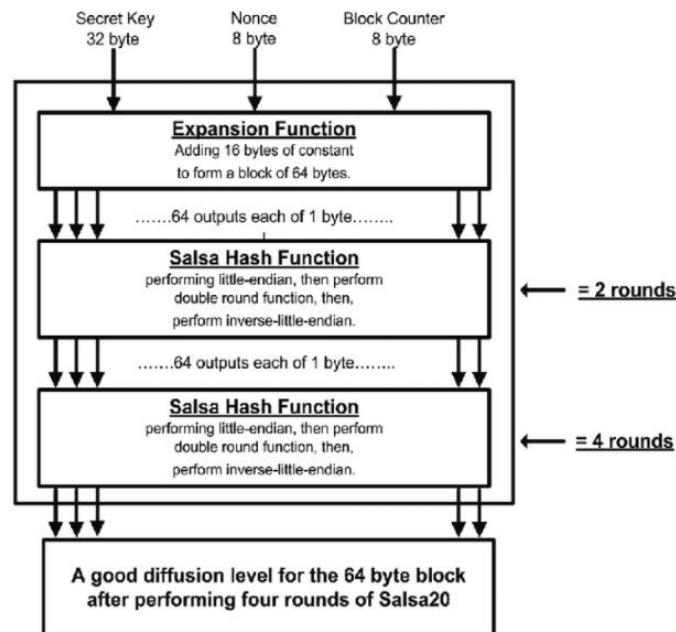


Figure 40 - Encryption by Stream Ciphers

performing the hash function (Bernstein, n.d.). Moreover, Figure 41 shows the general mechanism that stream cipher used to create ciphertext in its most basic form. Once the key is expanded it is XORed, in the case of Salsa20, against the bytes of the data. Note that more processes, such as rounds, and hashes can take place.

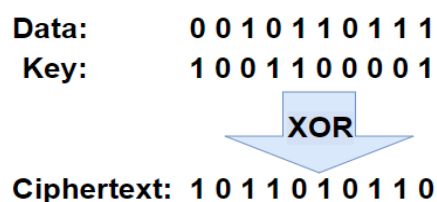


Figure 41 – Encryption by Stream Ciphers

This information contributed to the creation of a function to shift the encrypted bytes in the files by one at a time in a circular fashion. The encryption begins at every byte in the file which aims to find the original position at which the data was encrypted if the encryption started at an offset. A visual representation of the shift can be seen in Figure

42, in this case the decryption would begin in the first line, at the first character “T”, then at second character “h”, then at the third, “i” and so on throughout the entire file.

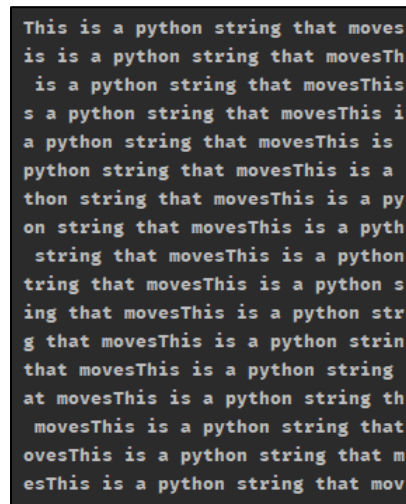


Figure 42 – Shifting Bytes to Find Position

The shifting function was looking for the header of the original file as a match. In this case the beginning of the file was “**ffd8ffe10018457869660000**”. Note that the magic bytes of a .JPG file are “**ffd8ff**”, however this resulted in some false positives, therefore a wider range of bytes was selected.

With this process the decryption was successful as shown in Figure 43. The beginning of the decrypted file starts at byte 65, and the first 64 bytes align the Salsa20 stream. The decryption begins at a specific location which is irrespective of the beginning of the file.

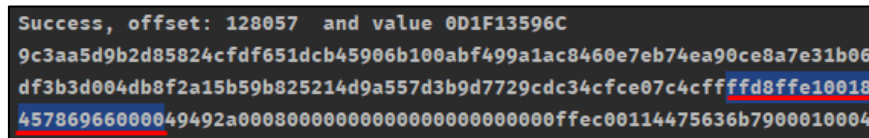


Figure 44 - File Header Found

Finally, all of the files for which the key and nonce pair were available were recovered using the information retrieved from the memory dumps. Figure 44 shows a byte by byte comparison of the decrypted files with the extracted keys and the original files before

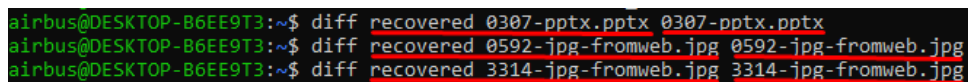


Figure 43 – Diff of Recovered and Original Files

they were encrypted. The diff command was used against them, and no differences were found. Figure 45 shows the encrypted and original files in the directory.

Name	Date	Type	Size
0307-pptx.pptx.gp7...	05/02/2022 16:40	GP7K2 File	3,078 KB
0592-jpg-fromweb.j...	05/02/2022 16:40	GP7K2 File	126 KB
3314-jpg-fromweb.j...	05/02/2022 16:41	GP7K2 File	3,442 KB
recovered_0307-ppt...	18/06/2014 14:23	Microsoft PowerP...	3,078 KB
recovered_0592-jpg...	10/02/2022 15:42	JPG File	126 KB
recovered_3314-jpg...	10/02/2022 15:42	JPG File	3,442 KB

Figure 45 - Sodinokibi Decrypted and Recovered

### 3.4.5 Experiment 1 - Evaluation

The initial experiment for Salsa20 key recovery from real ransomware, was successful in identifying and extracting the Salsa20 key and nonce pairs from the memory of a running ransomware process. The method and tool developed to extract Salsa20 key and nonce pairs from memory was successful in extracting those pairs from real cryptographic ransomware which had not been shown before using live memory forensics techniques. In this case, each of the multiple pairs found were matched to the key and nonce used by the ransomware to encrypt one file. The link between files and the pair was shown by finding the nonce in the metadata appended to the file by the ransomware, and then the associated key used to decrypt the file. This side channel attack on memory for recovery of the keys, could be used in practise to avoid paying ransom and recover users encrypted files if the memory captures were taken when the Sala keys were present in memory.

It was found that the files encrypted by this sample of Sodinokibi can be decrypted with the Salsa20 key and nonce pair, but decryption has to begin at 65 bytes into the Salsa20 encryption stream aligned with the beginning of the encrypted file. It is important to note that this decryption process does not seem documented in the available literature and that it was successfully used to decrypt all the files for which the key and nonce pair were recovered in the experiments.

This information was used to further develop the Sodinokibi Salsa20 extraction tools to relate encryption key and nonce pairs to the files and use the decryption information to decrypt files and strip the metadata appended by the ransomware. This provides an alternative decryption tool for this sample of Sodinokibi ransomware which until now was only available through Bitdefender (*Bitdefender Offers Free Universal Decryptor for REvil/Sodinokibi Ransomware*, n.d.) .

### 3.4.6 Experiment 2 - Sodiokibi Large Scale Key Recovery

The Salsa20 extraction tool was able to identify and extract encryption key and nonce pairs from the cryptographic ransomware sample while being executed. for small numbers of files. To be able to determine how many keys could be successfully extracted, and also to produce a timeline of when these might be memory, a much larger data set of 4000 files of mixed types were used in a similar experiment to the previous. After some

initial experimentation 10 second memory capture intervals starting at five seconds was decided on.

This process was automated by created a PowerShell script which allows a more rigorous way of taking snapshots at specified intervals, over the manual process used in the previous experiment. The script uses a PowerShell plugin to be able to interact with the VMWare VM and allows automation of pausing the VM and taking snapshots. It waits a certain number of seconds when it is executed, then it pauses the virtual machine and takes a snapshot, which then allows the memory to be extracted. This process is executed in a loop until the script is paused manually. This script is shown in Figure 46.

```
cd "C:\Program Files (x86)\VMware\VMware Workstation"

$RunningVMs = .\vmrun list | select-object -skip 1
$counter = 10
start-sleep -seconds 4;

while($true)
{
    Foreach ($RunningVMs in $RunningVMs)
    {
        "Suspending $RunningVMs..."
        .\vmrun suspend "$RunningVMs"
    }

    $counter++

    .\vmrun snapshot "C:\Users\Experiment\Documents\Virtual
Machines\W10Sodinokibi\W10Sodinokibi.vmx" $counter

    .\vmrun start "C:\Users\Experiment\Documents\Virtual
Machines\W10Sodinokibi\W10Sodinokibi.vmx"

    start-sleep -seconds 1;
}
```

Figure 46 – PowerShell Script

Figure 47 below, shows that the results of the experiment to find total numbers of keys found in memory, for same Sodinokibi sample on the larger data set of 4000 mixed files, with the memory captures taken at 10 seconds intervals. It was found that this strain of ransomware creates the ransom note around the same time as the keys start to be identified in memory, at around 15 seconds after execution, when over a thousand keys are identified in memory. Furthermore, it was found that the number of keys in memory increase for the next 3 intervals, over the next 30 seconds, peaking at 45 seconds from execution with a total of 1769 keys extracted at that point.

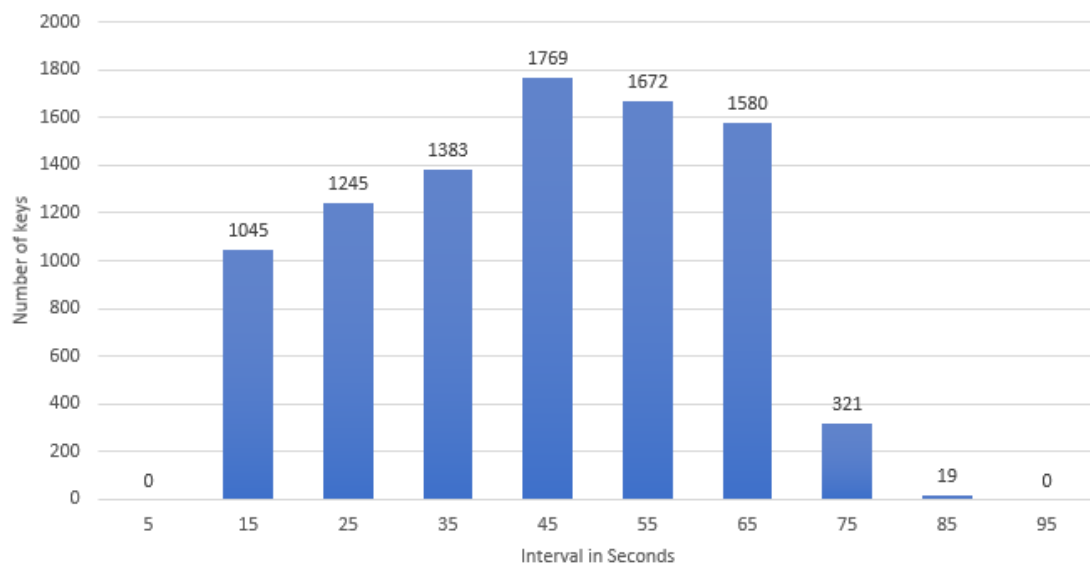


Figure 48 – Total Number of Keys in 10 Seconds Intervals

However, after some analysis of the extracted keys it was shown that more than 8000 Salsa20 keys were found over the experiment total time. After, comparing key and nonce values, it was observed that the keys among the memory snapshots were not unique as one key and nonce pair may appear over several different memory dumps. The numbers of unique keys were then analysed, as shown in Figure 48.

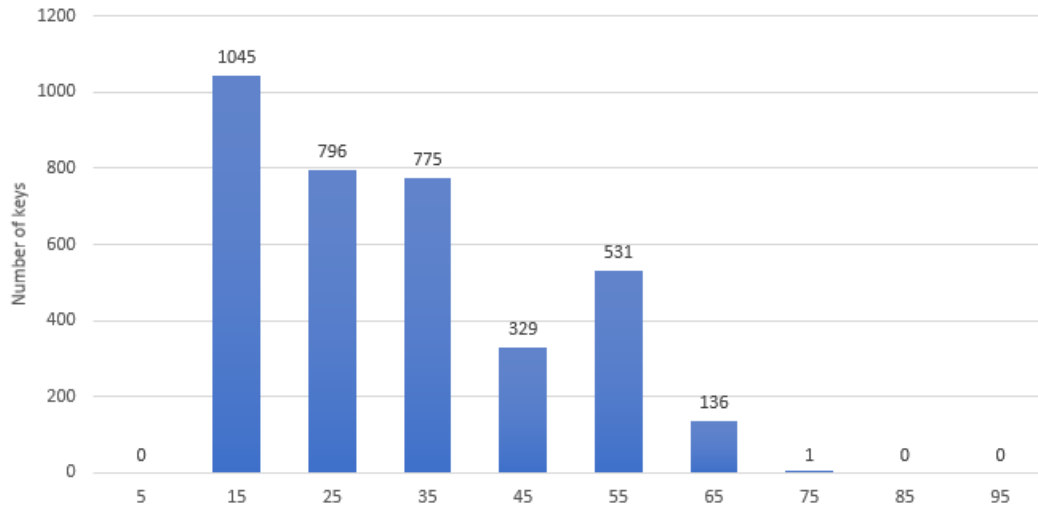


Figure 47 – Unique Keys in Memory in 10 Seconds Interval

Figure 48 shows the unique key and nonce pairs that pertain to each of the intervals. For instance, interval 25 has 796 pairs that are not found in any other memory dump. To do this calculation, the pairs found in each interval were compared to all the previous intervals. If a pair was found in a previous interval, it was removed from the list. The number of unique pairs recovered were 3612 out of 4000 encrypted files, which is over 90% of the keys recovered. Furthermore, Figure 48 also shows that Sodinokibi initialises a large amount of the keys in the first interval, it does not gradually increase the numbers

as might be expected. This could be due to the malware's multi-threaded behaviour for the key creation and encryption perhaps. It also initialises a lower number of pairs in the following intervals; however, interval 55 has an increased number of new unique pairs when compared to 45. This could be due to attempts by the malware to delete keys from memory once the file encryption has been completed.

Figure 49 below, attempts to show the persistence of keys in memory. It shows the total amounts of keys at each interval and highlights the numbers of unique new keys created at each interval and the numbers of keys still in memory from the previous captures. To calculate the presence of the unique pairs across the memory dumps, the total unique pairs in each memory dump were subtracted from the number of unique pairs found in each previous dumps and added to a list.

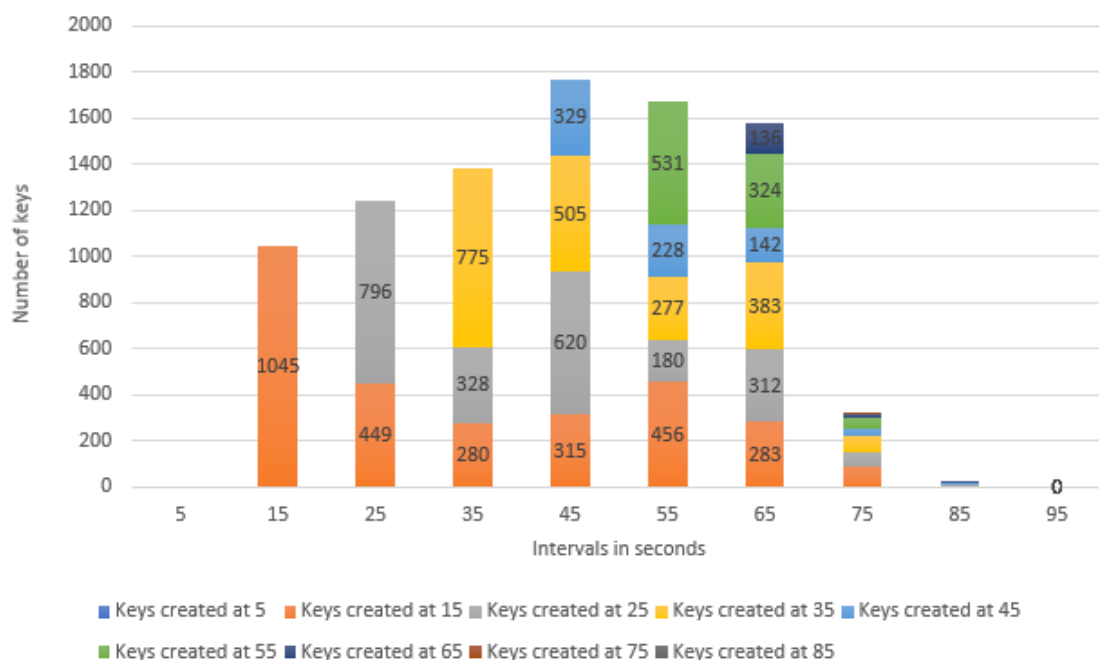


Figure 49 – Persistence of Keys in Memory

Showing how long the pairs found at each memory dump are present in memory is interesting and shows perhaps the pattern of chunks of keys being created and then gradually deleted after use. For instance, out of 1045 unique pairs found at 15 seconds, 449 of them were shown still to be in memory at 25 seconds, and 280 at 35 seconds. An anomalous pattern is observed among keys from intervals 15, 25, and 35 seconds which is the same as can be seen in Figure 48 in that the number of keys decreases but it increases before dropping again. Perhaps this could indicate swapping of the memory from RAM to disk and back, but more experimentation and analysis might be interesting to explore this.

This experiment showed that out of 4000 files encrypted by this sample of cryptographic ransomware 90.3% were recovered using live memory forensics techniques at 10 second

intervals. This is promising towards a mitigation method, as if the memory could be captured frequently possibly all keys could be captured, or perhaps a trigger such as hooking API function calls might be used to initiate the captures. It also shows that the encryption of the files by this sample of Sodinokibi begins at around second 15 with the described setup. This seemed fairly constant for the various initial experiments carried out. Furthermore, it shows that the amount of time that the key and nonce pair last in memory is between around 10 seconds and 60 seconds from the start of the ransomware execution. If this period could be monitored the keys are all going to be in memory for this small time period and could be captured in theory.

Lastly, a pattern is observed which suggests that some keys are not found in memory at a certain interval, instead they are loaded again, then they disappear again for the last time. This could be due to either the ransomware key management actively deleting keys or by the virtual memory of the operating system running out, and therefore using disk space to store those keys perhaps.

### 3.4.8 Experiment 3 – Sodinokibi Encryption for Key Time Lining

Another experiment was performed to attempt to analyse the time line of keys created in memory for the sample ransomware. It was carried with a smaller data set of 20 files but this time of exactly the same size and with snapshots taken at 1 second intervals. This was to attempt to identify keys being created and destroyed as encryption was carried out, so files were used of the same size. This experiment was performed 6 times and on only one occasion the 20 key and nonce pairs were found. For the rest, no more than 4 keys were found. The files that were encrypted were ordered by modified date which is a parameter that this ransomware changes after encrypting a file as seen in Figure 50, on the right the original file and on the left the same file after encryption.

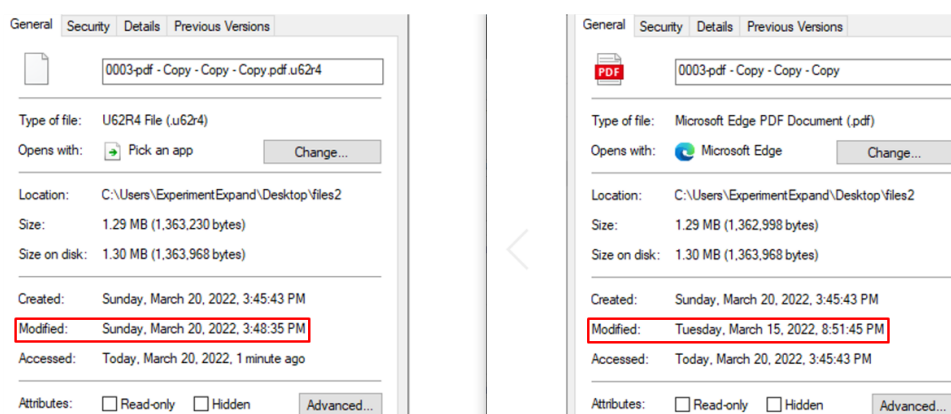


Figure 50 - Modified Time After Encryption

Figure 51 shows the results of the experiment where all 20 keys were found, with the files being encrypted on the x axis, and the order sorted by the modified date. Ordering by modified timestamp shows the order the files were encrypted in. The figure shows all of

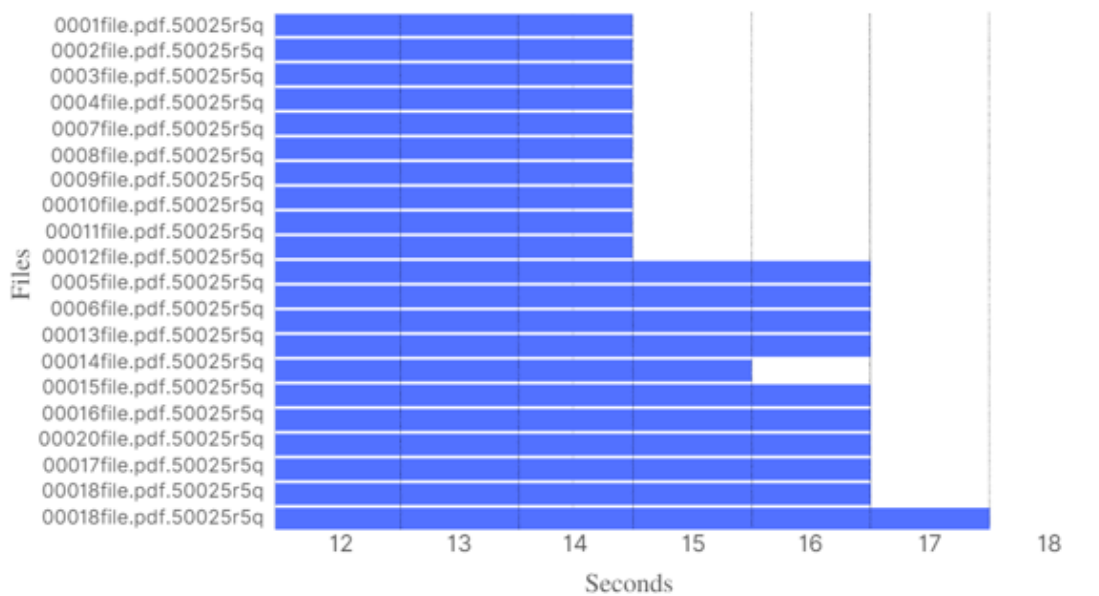


Figure 51 - Encryption Key Timeline

the keys are initialised at second 12, at the same time that the ransom note was created. So, nothing is found for the first 11 seconds similar to the previous experiments. For the next 3 seconds all 20 keys can be found in memory. At 15 seconds the first key and nonce pairs for the first 10 files were not found and may have been removed from memory by the ransomware, after the encrypted using those keys has been completed. 3 seconds later the remaining keys were not found.

In this last experiment the key and nonce pair were recovered for the 20 files that were encrypted, although it had to be run 6 times, indicating that the time at which the memory dump is captured is critical for small number of files. Furthermore, it also showed that encryption begins after the ransom note has been created by the ransomware at 12 seconds which aligns to the documented behaviour. Moreover, the order in which encryption keys disappear from memory is the same in which the files are encrypted, pointing towards that the key nonce pair of the first encrypted file seem to be removed from memory first. Also, the encryption of files appears to happen in bulk as well as the creation and removal of key and nonce pairs. In this experiment, the keys that were found in memory for the longest period were identified for 5 seconds, as opposed to the first 10 files pairs which were only seem in memory for 2 seconds.

### 3.5 Conclusion

This chapter begins by presenting the method that was derived from the literature review. The method builds upon previous work leveraging memory forensics tools against cryptographic ransomware. Furthermore, this method targets gaps in the literature that do not cover unique key per file encryption, as well as new ciphers such as Salsa20 being



used by recent ransomware and the development of new tools that can identify the encryption keys of those new ciphers.

Moreover, there are two main parts to the work. The first aims to reproduce the previous work on live memory forensics against ransomware identifying AES keys in memory dumps (Davies et al., 2020) and to attempt to create a similar method for extraction of the commonly used Salsa20 encryption keys. This was successful after a series of experiments building one by one towards the final goal. A new method and a tool for Salsa key and nonce extraction from memory were developed.

The second describes the application for real ransomware of the Salsa20 method of key extraction from the volatile memory of a virtual machine that has ransomware being executed. To run this type of malware in a secure manner a safe environment is needed and therefore the Environment was designed detailing the steps taken to ensure the ransomware sample does not accidentally infect or damage any other system or process other than the test files. This was successfully used to carry out experiments with real Sodinokibi ransomware without any incident. Furthermore, the cryptographic ransomware was selected from the literature due to its type and recent use, to perform a side channel attack and extract encryption keys and decrypt files without paying the ransom. The results of the experiments and their evaluation showed that it is possible to leverage live memory forensics techniques to mitigate ransomware as previously done by Davies et al (2020). The results further showed that it is possible to extract many Salsa20 keys from the volatile memory, with up to around 90% being recovered in one set of experiments. They also describe a process of decrypting Sodinokibi files and a timeline of the Salsa20 key and nonce pairs in memory.

It is important to note that all the experiments that involved running the ransomware were carried out in a virtual machine with the same resources specified previously. It is likely that if the resources were increased, the window for finding the key and nonce pair would possibly be reduced, but further experiments with a variety of different resources, and using a large data set of files would be interesting. It may be that more resources such as numbers of cores and memory size, may allow for faster performance of the ransomware, for instance a higher number of encryption threads, and therefore higher overall encryption speed (Ravikant & Alexander, 2021). However, a typical machine would have many other processes running and perhaps the ransomware would not have as much overall resources to use, so further experiments with a more realistic environment may also be useful.

## 4. Evaluation

### 4.1 Introduction

The use of live memory forensics to mitigate cryptographic ransomware is a technique that only limited amount of research work has been done, and while there are other reactive techniques, they have their own drawbacks. Davies et al (2020), Bajpai (2020), and Yuste and Pastrana (2021) were successful in extracting cryptographic keys to decrypt files, and the methods seem promising, but they did not try to recover unique key per file for ransomware encrypting with multiple keys. Furthermore, AES and RSA keys were recovered in the related work, but there seems to be a recent shift in the ciphers used by ransomware developers for the victim file encryption. This shift seems to favour Salsa20 stream cipher mainly with recent ransomware. This work was based around building on the previous research towards Salsa20 key recovery for ransomware mitigation. The work has been able to recover these unique per key file for Salsa20 and decrypt ransomware encrypted files. However, the use of live memory forensics techniques could possibly not be used to mitigate a real ransomware infection completely at this point as the volatile memory needs to be captured at a certain interval when all of the keys are in memory. However, if the memory could be monitored at the right time periods the keys could be recovered with the method developed, and it may be possible to mitigate real ransomware attacks.

### 4.2 Aims and Objectives

This section aims to critically discuss this work by aligning the original objectives with the findings. The objectives and findings are also related to the consulted literature to compare the results and general context.

#### 4.2.1 Objective One – Literature review

The first objective was the literature review around current ransomware cryptographic management, ransomware detection and recovery and live memory forensics methods. The outcome of the literature review is discussed in section 2.6, which describes the classification of cryptographic ransomware and the reason for this type of ransomware to be found the most damaging. The literature review also discusses the detection and reactive mechanisms, which seem to be the two major approaches to mitigate cryptographic ransomware. Detection mechanisms are mostly based on machine learning (Berrueta et al., 2019) and are generally prevalent to reactive mechanisms. Furthermore, new emerging reactive mechanisms include the use of live forensics techniques to mitigate ransomware (Bajpai, 2020; Davies et al., 2020; Yuste & Pastrana, 2021). However, they have been unable to recover files that were encrypted with multiple encryption keys. The last chapter of the literature review found that there seems to be a

shift in the selected ciphers used by cryptographic ransomware, with Salsa20 being the predominant type and the work was aimed towards this area.

#### 4.2.2 Objective Two – Design and Implementation

Objective two consisted in designing experiments to evaluate forensics techniques and find encryption keys in the volatile memory and verify such findings. The design of the experiments is divided into two. First, a design was developed to replicate and verify the method used by Davies et al (2020) to find encryption keys in memory, specifically AES keys. This design was used to implement the initial experiments that were successful in extracting AES keys from the volatile memory and decrypt files. The second design was similar to the first, but it considered the safety measures that have to be taken to execute real ransomware samples. The automatization of the process of capturing the memory of the virtual machine was achieved by using a PowerShell script which was precise when taking snapshots and reduced workload. This design was implemented through the experiments which were successful at developing a tool to extract Salsa20 key and nonce pairs from memory. This tool was also successful against cryptographic ransomware to recover multiple unique pairs used to encrypt a file each. Furthermore, the verification of the findings was done through the decryption of files encrypted by a sample of Sodinokibi with the recovered key and nonce pairs. Using the same design, a timeline of the memory of the sample was created identifying how many keys, for how long and when they were in the volatile memory. Lastly, the experiments also showed that the order of encryption of the files is the order in which the keys disappear from memory. The key and nonce pair used to encrypt the first file is the first pair to disappear from memory, this seems to happen in bulk and in order for other files. However, it is likely that if the resources given to the virtual machine are increased, the window for key extraction would become narrower. It is important to note that previous work done in this field were unable to recover per file encryption key and nonce pairs and that they focused on extracting AES and RSA keys (Bajpai, 2020; Davies et al., 2020; Yuste & Pastrana, 2021) for which there is a wealth of literature (Halderman et al., 2009; Hargreaves & Chivers, 2008; Kaplan & Geiger, 2007; Maartmann-Moe et al., 2009) so the work builds on these, and adds a new method for Salsa20 key extraction and then applies this towards real ransomware mitigation.

#### 4.2.3 Objective Three - Evaluation

Objective three is the evaluation of the project objectives. This project built upon current research on the field of memory forensics to mitigate ransomware, extending the field to finding another type of cipher, in this case the stream cipher Salsa20, and in retrieving multiple encryption key and nonce pairs used to encrypt multiple files. Furthermore, it showed that a side channel attack can be performed on recent cryptographic ransomware to recover the compromised data. These findings are consistent with findings from other

literature (Bajpai, 2020; Davies et al., 2020; Yuste & Pastrana, 2021) and add a new method. Moreover, this work was able to produce a decryption tool for the sample of Sodinokibi selected. The Salsa20 extraction tool was extended to search through encrypted files to find those for which the nonce is recovered, to decrypt those files, remove the ransomware's metadata and place them in a new folder. This is also a new method not documented anywhere the author could find. An important part of the project was the design of the environment on which the experiments were carried out. This design proved useful in reducing manual workload as it was semi automatised. The virtual machine was paused, and the memory captured at specified intervals without the interaction of the researcher. This allowed for more experiments to be conducted, than if the process was done manually.

There are some limitations found to be related to the use of memory forensics. Using a virtual machine to execute a sample of ransomware enables the ability to restrain resources that the malware has access to which is beneficial as a proof-of-concept. However, this method would require more dynamism to be applied to the real world such as that discussed by Bajpai (2020).. Additionally, more experiments could have been carried out if more time was available, which would also have produced clearer results, such as an average of data recovery after infection. Also, a more realistic scenario in which the virtual machine was given more resources, and other typical processes running could have been looked into. Lastly, the decryption of cryptographic ransomware needs extra research. Even if the encryption key and nonce pair have been successfully extracted, ransomware developers use different ways of encrypting files which poses a challenge of its own.

## 4.2 Self Appraisal

The impact of the findings of this project could go beyond the discussed fields and into other areas related to cryptographic ciphers such as Salsa20 and its variants - which is in itself an unexpected achievement. Furthermore, it has proven to be a novel way when used against ransomware. The fact that it is possible to extract multiple encryption keys from the volatile memory to recover files could have a beneficial impact in the fight against cryptographic ransomware.

The automation of the experiments allowed the increase of the number of experiments. Furthermore, the environment in which the ransomware was executed proved to be flexible and safe enough to attain good results without adverse effects. However, the total allocated time to the project could be considered a downside as it prevented further experiments from being carried out.

Overall, the project was found to have successful findings by the researcher. However, if it had to be repeated the time allocated to the experiments would have been expanded.

Furthermore, the use of an alternative tool such as LaTeX would have been considered since the time spent formatting and reformatting the document could have been used in other tasks. Additionally, the project could have been broken down into smaller projects such as focusing on identifying and developing a tool to extract multiple types of keys in memory. A second project would have used that tool against cryptographic ransomware and recover the files.

## 4.3 Future Work

Currently, a single solution to mitigate cryptographic ransomware does not exist as discussed in section 2.3.1. However, there are some advances in the struggle against this malware. This work has explored some of them and it has contributed to it. In this section there are some areas of research related to this work that could be beneficial to investigate in the future.

Real time monitoring of the volatile memory of a host enables a centralized solution for capturing encryption keys that can be used to recover files from ransomware. This solution would merge the use of live memory forensics and real time technologies to mitigate ransomware. While there is a version of this solution (Bajpai, 2020), it does not address the problem of using a key for each file encrypted or the use of different ciphers to encrypt data.

The timeline discussed in experiments 4 and 5 could help to develop new mechanisms to extract important information from the memory of an infected host. New strains of cryptographic ransomware could be analysed with this technique. Furthermore, a better understanding of how new ransomware is designed can be achieved through the documentation of the timeline.

The development of the identification and extraction of Salsa20 tool has proved very useful in this work – the base of the project. Therefore, additional tools could be developed to extract other types of ciphers, such as Chacha20, that may be used by other strains of ransomware (Yuste & Pastrana, 2021).

A step between capturing a snapshot to be analysed and real time monitoring would be the use of a solution that allows capturing the memory of a physical computer in an easy manner. This way a snapshot of the system could be taken when ransomware is detected and analysed later to extract the keys. At present, such technique does not seem to exist.

## References

- A deep dive into Phobos ransomware - Malwarebytes Labs / Malwarebytes Labs.* (n.d.). Retrieved November 15, 2021, from <https://blog.malwarebytes.com/threat-analysis/2019/07/a-deep-dive-into-phobos-ransomware/>
- A Detailed Walkthrough of Ranzy Locker Ransomware TTPs.* (n.d.). Retrieved March 30, 2022, from <https://www.picussecurity.com/resource/blog/a-detailed-walkthrough-of-ranzy-locker-ransomware-ttps>
- Adamov, A., & Carlsson, A. (2017). The state of ransomware. Trends and mitigation techniques. *Proceedings of 2017 IEEE East-West Design and Test Symposium, EWDTs 2017*. <https://doi.org/10.1109/EWDTs.2017.8110056>
- Aidan, J. S., Verma, H. K., & Awasthi, L. K. (2018). Comprehensive survey on petya ransomware attack. *Proceedings - 2017 International Conference on Next Generation Computing and Information Systems, ICNGCIS 2017*, 131–136. <https://doi.org/10.1109/ICNGCIS.2017.30>
- Akbanov, M., Vassilakis, V. G., & Logothetis, M. D. (2019). WannaCry Ransomware: Analysis of Infection, Persistence, Recovery Prevention and Propagation Mechanisms. *Journal of Telecommunications and Information Technology, nr 1*(1), 113–124. <https://doi.org/10.26636/JTIT.2019.130218>
- All your Private Keys are Belong to Us.* (n.d.). Retrieved December 2, 2021, from <https://www.trapkit.de/articles/all-your-private-keys-are-belong-to-us/>
- Al-rimy, B. A. S., Maarof, M. A., & Shaid, S. Z. M. (2018). Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Computers and Security, 74*, 144–166. <https://doi.org/10.1016/J.COSE.2018.01.001>
- AMOSSYS Security Blog.* (n.d.). Retrieved March 4, 2022, from <https://blog.amossys.fr/sodinokibi-malware-analysis.html>
- Bajpai, P. (2020). *EXTRACTING RANSOMWARE'S KEYS BY UTILIZING MEMORY FORENSICS*.
- Bajpai, P. , R., Sood, A. K., & Enbody (2018). A key-management-based taxonomy for ransomware. *ECrime Researchers Summit, ECrime, 2018-May*, 1–12. <https://doi.org/10.1109/ECRIME.2018.8376213>
- Beaman, C., Barkworth, A., Akande, T. D., Hakak, S., & Khan, M. K. (2021). Ransomware: Recent advances, analysis, challenges and future research directions. *Computers & Security, 111*, 102490. <https://doi.org/10.1016/j.cose.2021.102490>

- Berkay Celik, Z., Mcdaniel, P., & Bowen, T. (2018). Malware modeling and experimentation through parameterized behavior. *Journal of Defense Modeling and Simulation: Applications, Methodology*, 15(1), 2017. <https://doi.org/10.1177/1548512917721755>
- Bernstein, D. J. (n.d.). *Salsa20 specification*.
- Berrueta, E., Morato, D., Magana, E., & Izal, M. (2019). A Survey on Detection Techniques for Cryptographic Ransomware [Article]. *IEEE Access*, 7, 1–1. <https://doi.org/10.1109/ACCESS.2019.2945839>
- Bitdefender Offers Free Universal Decryptor for REvil/Sodinokibi Ransomware*. (n.d.). Retrieved March 4, 2022, from <https://www.bitdefender.com/blog/labs/bitdefender-offers-free-universal-decryptor-for-revil-sodinokibi-ransomware/>
- BlackMatter Ransomware Analysis; The Dark Side Returns | McAfee Blog*. (n.d.). Retrieved March 30, 2022, from <https://www.mcafee.com/blogs/enterprise/mcafee-enterprise-atr/blackmatter-ransomware-analysis-the-dark-side-returns/>
- Cohen, A., & Nissim, N. (2018). Trusted detection of ransomware in a private cloud using machine learning methods leveraging meta-features from volatile memory [Article]. *Expert Systems with Applications*, 102, 158–178. <https://doi.org/10.1016/j.eswa.2018.02.039>
- Continella, A., Guagnelli, A., Zingaro, G., de Pasquale, G., Barengi, A., Zanero, S., & Maggi, F. (2016). ShieldFS: A self-healing, ransomware-aware file system. *ACM International Conference Proceeding Series, 5-9-December-2016*, 336–347. <https://doi.org/10.1145/2991079.2991110>
- CoronaVirus Ransomware*. (n.d.). Retrieved October 6, 2021, from <https://www.cyberark.com/resources/threat-research-blog/coronavirus-ransomware>
- Craciun, V. C., Mogage, A., & Simion, E. (2019). Trends in design of ransomware viruses. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11359 LNCS, 259–272. [https://doi.org/10.1007/978-3-030-12942-2\\_20](https://doi.org/10.1007/978-3-030-12942-2_20)
- CryptoWall Ransomware Threat Analysis | Secureworks*. (n.d.). Retrieved December 23, 2021, from <https://www.secureworks.com/research/cryptowall-ransomware>
- Darkside Ransomware | Chuong Dong*. (n.d.). Retrieved March 28, 2022, from <https://chuongdong.com/reverse%20engineering/2021/05/06/DarksideRansomware/>
- Davies, S. R., Macfarlane, R., & Buchanan, W. J. (2020). Evaluation of live forensic techniques in ransomware attack mitigation [Article]. *Forensic Science*

- International: Digital Investigation*, 33, 300979.  
<https://doi.org/10.1016/j.fsidi.2020.300979>
- Davies, S. R., Macfarlane, R., & Buchanan, W. J. (2021a). Exploring the Need for an Updated Mixed File Research Data Set. *7th International Conference on Engineering and Emerging Technologies, ICEET 2021*.  
<https://doi.org/10.1109/ICEET53442.2021.9659618>
- Davies, S. R., Macfarlane, R., & Buchanan, W. J. (2021b). Differential area analysis for ransomware attack detection within mixed file datasets. *Computers & Security*, 108, 102377. <https://doi.org/10.1016/J.COSE.2021.102377>
- de Gaspari, F., Hitaj, D., Pagnotta, G., de Carli, L., & Mancini, L. v. (2019). The Naked Sun: Malicious Cooperation Between Benign-Looking Processes. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12147 LNCS, 254–274.  
<https://doi.org/10.48550/arxiv.1911.02423>
- EternalPetya and the lost Salsa20 key* | *Malwarebytes Labs*. (n.d.). Retrieved March 29, 2022, from <https://blog.malwarebytes.com/threat-analysis/2017/06/eternalpetya-lost-salsa20-key/>
- Fayi, S. Y. A. (2018). What Petya/NotPetya Ransomware Is and What Its Remediations Are. *Advances in Intelligent Systems and Computing*, 738, 93–100.  
[https://doi.org/10.1007/978-3-319-77028-4\\_15](https://doi.org/10.1007/978-3-319-77028-4_15)
- GandCrab V4.0 Analysis: New Shell, Same Old Menace*. (n.d.). Retrieved March 12, 2022, from <https://www.fortinet.com/blog/threat-research/gandcrab-v4-0-analysis--new-shell--same-old-menace>
- Genç, Z. A. (2020). *ANALYSIS, DETECTION, AND PREVENTION OF CRYPTOGRAPHIC RANSOMWARE* Dr Gianluca STRINGHINI.
- Genç, Z. A., Lenzini, G., & Ryan, P. Y. A. (2018). No Random, No Ransom: A Key to Stop Cryptographic Ransomware. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10885 LNCS, 234–255. [https://doi.org/10.1007/978-3-319-93411-2\\_11](https://doi.org/10.1007/978-3-319-93411-2_11)
- Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., Feldman, A. J., Appelbaum, J., & Felten, E. W. (2009). Lest we remember. *Communications of the ACM*, 52(5), 91–98.  
<https://doi.org/10.1145/1506409.1506429>



- Hargreaves, C., & Chivers, H. (2008). Recovery of encryption keys from memory using a linear scan. *ARES 2008 - 3rd International Conference on Availability, Security, and Reliability, Proceedings*, 1369–1376. <https://doi.org/10.1109/ARES.2008.109>
- Hassan, N. A. (2019). Ransomware Families. *Ransomware Revealed*, 47–68. [https://doi.org/10.1007/978-1-4842-4255-1\\_3](https://doi.org/10.1007/978-1-4842-4255-1_3)
- Hsiao, S. C., & Kao, D. Y. (2018). The static analysis of WannaCry ransomware. *International Conference on Advanced Communication Technology, ICACT, 2018-February*, 153–158. <https://doi.org/10.23919/ICACT.2018.8323680>
- Humayun, M., Jhanjhi, N. Z., Alsayat, A., & Ponnusamy, V. (2021a). Internet of things and ransomware: Evolution, mitigation and prevention. *Egyptian Informatics Journal*, 22(1), 105–117. <https://doi.org/10.1016/J.EIJ.2020.05.003>
- Humayun, M., Jhanjhi, N. Z., Alsayat, A., & Ponnusamy, V. (2021b). Internet of things and ransomware: Evolution, mitigation and prevention. *Egyptian Informatics Journal*, 22(1), 105–117. <https://doi.org/10.1016/J.EIJ.2020.05.003>
- Immediate action required to avoid Ransomware pandemic - INTERPOL*. (n.d.). Retrieved October 6, 2021, from <https://www.interpol.int/en/News-and-Events/News/2021/Immediate-action-required-to-avoid-Ransomware-pandemic-INTERPOL>
- Kaplan, B., & Geiger, M. (2007). *RAM is Key Extracting Disk Encryption Keys From Volatile Memory Thesis Report*.
- Kim, H. E., Yoo, D., Kang, J. S., & Yeom, Y. (2017). Dynamic ransomware protection using deterministic random bit generator. *2017 IEEE Conference on Applications, Information and Network Security, AINS 2017, 2018-January*, 64–68. <https://doi.org/10.1109/AINS.2017.8270426>
- Klein, T. (2006). *All your private keys are belong to us Extracting RSA private keys and certificates out of the process memory*. 1–7. <https://www.semanticscholar.org/paper/All-your-private-keys-are-belong-to-us-Extracting-Klein/cf85042cca0da125b860db7c2fefb38012396cbc>
- Kolodenker, E., Koch, W., Stringhini, G., & Egele, M. (2017). PayBreak : Defense against cryptographic ransomware. *ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*, 599–611. <https://doi.org/10.1145/3052973.3053035>
- Lemmou, Y., & Souidi, E. M. (2018). Inside GandCrab Ransomware. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11124 LNCS, 154–174. [https://doi.org/10.1007/978-3-030-00434-7\\_8](https://doi.org/10.1007/978-3-030-00434-7_8)

- Maartmann-Moe, C., Thorkildsen, S. E., & André Årnes. (2009). The persistence of memory: Forensic identification and extraction of cryptographic keys. *Digital Investigation*, 6(SUPPL.), S132–S140. <https://doi.org/10.1016/J.DIIN.2009.06.002>
- Makrakis, G. M., Koliass, C., Kambourakis, G., Rieger, C., & Benjamin, J. (2021). Industrial and Critical Infrastructure Security: Technical Analysis of Real-Life Security Incidents. *IEEE Access*, 9, 165295–165325. <https://doi.org/10.1109/ACCESS.2021.3133348>
- Mehnaz, S., Mudgerikar, A., & Bertino, E. (2018). RWGuard: A Real-Time Detection System Against Cryptographic Ransomware. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11050 LNCS, 114–136. [https://doi.org/10.1007/978-3-030-00470-5\\_6](https://doi.org/10.1007/978-3-030-00470-5_6)
- Mohammad, A. H. (2020). Analysis of Ransomware on Windows platform COMPARING TWO FEATURE SELECTIONS METHODS (INFORMATION GAIN AND GAIN RATIO) ON THREE DIFFERENT CLASSIFICATION ALGORITHMS USING ARABIC DATASET. View project Analysis of Ransomware on Windows platform. *IJCSNS International Journal of Computer Science and Network Security*, 20(6). <https://doi.org/10.13140/RG.2.2.11150.59202>
- Moussaileb, R., & le Boudier, H. (2021). 17 A Survey on Windows-based Ransomware Taxonomy and Detection Mechanisms: Case Closed? <https://doi.org/10.1145/3453153>
- Nakano, Y., Basu, A., Kiyomoto, S., & Miyake, Y. (2014). Key Extraction Attack Using Statistical Analysis of Memory Dump Data. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8924, 239–246. [https://doi.org/10.1007/978-3-319-17127-2\\_17](https://doi.org/10.1007/978-3-319-17127-2_17)
- Nissim, N., Lahav, O., Cohen, A., Elovici, Y., & Rokach, L. (2019). Volatile memory analysis using the MinHash method for efficient and secured detection of malware in private cloud. *Computers & Security*, 87, 101590. <https://doi.org/10.1016/J.COSE.2019.101590>
- Oz, H., Aris, A., Levi, A., & Uluagac, A. S. (2021). A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions. *ACM Computing Surveys*, 1(1). <http://arxiv.org/abs/2102.06249>
- Phobos Ransomware, A Combo Of CrySiS & Dharma*. (n.d.). Retrieved November 15, 2021, from <https://www.coveware.com/blog/phobos-ransomware-distributed-dharma-crew>
- Poudyal, S. (2021). *Multi-level analysis of Malware using Machine Learning*.

- Raffetseder, T., Kruegel, C., & Kirda, E. (2007). Detecting System Emulators. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4779 LNCS, 1–18. [https://doi.org/10.1007/978-3-540-75496-1\\_1](https://doi.org/10.1007/978-3-540-75496-1_1)
- Ramsdell, K. A. W., & Esbeck, K. E. (2021). *EVOLUTION OF RANSOMWARE*.
- Ravikant, T., & Alexander, K. (2021). No Title أنظمة أتمتة المصانع. *Taking Deep Drive into Sodinokibi Ransomware*, 1–31. <https://emea.mitsubishielectric.com/ar/products-solutions/factory-automation/index.html>
- Rehman, H. ur, Yafi, E., Nazir, M., & Mustafa, K. (2018). Security Assurance Against Cybercrime Ransomware. *Advances in Intelligent Systems and Computing*, 866, 21–34. [https://doi.org/10.1007/978-3-030-00979-3\\_3](https://doi.org/10.1007/978-3-030-00979-3_3)
- Return of the Darkside: Analysis of a Large-Scale Data Theft Campaign*. (n.d.). Retrieved March 29, 2022, from <https://www.varonis.com/blog/darkside-ransomware>
- Reza, S. M. S., Arifeen, M. M., Tiong, S. K., Akhteruzzaman, M., Amin, N., Shakeri, M., Ayob, A., & Hussain, A. (2020). Salsa20 based lightweight security scheme for smart meter communication in smart grid. *Telkomnika (Telecommunication Computing Electronics and Control)*, 18(1), 228–233. <https://doi.org/10.12928/TELKOMNIKA.V18I1.14798>
- Rossow, C., Dietrich, C. J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., Bos, H., & van Steen, M. (2012). Prudent practices for designing malware experiments: Status quo and outlook. *Proceedings - IEEE Symposium on Security and Privacy*, 65–79. <https://doi.org/10.1109/SP.2012.14>
- Salsa20 — PyCryptodome 3.14.1 documentation*. (n.d.). Retrieved April 3, 2022, from <https://pycryptodome.readthedocs.io/en/latest/src/cipher/salsa20.html>
- Salsa20 block diagram. / Download Scientific Diagram*. (n.d.). Retrieved March 31, 2022, from [https://www.researchgate.net/figure/Salsa20-block-diagram\\_fig4\\_280028128](https://www.researchgate.net/figure/Salsa20-block-diagram_fig4_280028128)
- Shannon, C. E. (1948). *A mathematical theory of communication / Nokia Bell Labs Journals & Magazine / IEEE Xplore*. A Mathematical Theory of Communication. <https://ieeexplore.ieee.org/document/6773024>
- Shamir, A., & Van Someren, N. (1999). Playing ‘Hide and Seek’ with Stored Keys. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1648, 118–124. [https://doi.org/10.1007/3-540-48390-X\\_9](https://doi.org/10.1007/3-540-48390-X_9)

- Sharif, S. O., & Mansoor, S. P. (2010). Performance analysis of stream and block cipher algorithms. *ICACTE 2010 - 2010 3rd International Conference on Advanced Computer Theory and Engineering, Proceedings*, 1. <https://doi.org/10.1109/ICACTE.2010.5578961>
- Shukla, M., Manjunath, S., Saxena, R., Mondal, S., & Lodha, S. (2015). WinOver enterprise dark data. *Proceedings of the ACM Conference on Computer and Communications Security, 2015-October*, 1674–1676. <https://doi.org/10.1145/2810103.2810131>
- Sikorski, Michael., & Honig, Andrew. (2012). *Practical Malware Analysis : a Hands-On Guide to Dissecting Malicious Software*. 802. [https://books.google.com/books/about/Practical\\_Malware\\_Analysis.html?id=FQC8EPYy834C](https://books.google.com/books/about/Practical_Malware_Analysis.html?id=FQC8EPYy834C)
- Sodinokibi Ransomware Is Storming Organizations*. (n.d.). Retrieved November 6, 2021, from <https://www.virsec.com/blog/following-in-gandcrabs-big-footsteps-sodinokibi-ransomware-is-storming-organizations>
- The State of CryptoWall in 2018*. (n.d.). Retrieved December 23, 2021, from <https://www.varonis.com/blog/cryptowall/>
- Threat Spotlight: Sodinokibi Ransomware*. (n.d.). Retrieved March 28, 2022, from <https://blogs.blackberry.com/en/2019/07/threat-spotlight-sodinokibi-ransomware>
- Threat Thursday: Karma Ransomware*. (n.d.). Retrieved March 30, 2022, from <https://blogs.blackberry.com/en/2021/11/threat-thursday-karma-ransomware>
- Usharani, S., Bala, P. M., & Mary, M. M. J. (2021). Dynamic Analysis on Crypto-ransomware by using Machine Learning: GandCrab Ransomware. *Journal of Physics: Conference Series*, 1717(1), 012024. <https://doi.org/10.1088/1742-6596/1717/1/012024>
- What We Know About Darkside Ransomware and the US Pipeline Attack*. (n.d.). Retrieved March 29, 2022, from [https://www.trendmicro.com/en\\_gb/research/21/e/what-we-know-about-darkside-ransomware-and-the-us-pipeline-attac.html](https://www.trendmicro.com/en_gb/research/21/e/what-we-know-about-darkside-ransomware-and-the-us-pipeline-attac.html)
- Yuste, J., & Pastrana, S. (2021). Avaddon ransomware: An in-depth analysis and decryption of infected systems. *Computers and Security*, 109. <https://doi.org/10.1016/J.COSE.2021.102388>
- Zimba, A., Wang, Z., & Chishimba, M. (2021). Addressing Crypto-Ransomware Attacks: Before You Decide whether To-Pay or Not-To [Article]. *The Journal of Computer Information Systems*, 61(1), 53–63. <https://doi.org/10.1080/08874417.2018.1564633>

세 훈이, 병 철윤, 소 람김, 기 윤김, 영 주이, 대 윤김, 해 룡박, 종 성김,  
금융정보보안학과국민대학교, Lee, S., Youn, B., Kim, S., Kim, G., Lee, Y., Kim,  
D., Park, H., Kim, J., & 서 론 I. (2019). A Study on Encryption Process and  
Decryption of Ransomware in 2019. *Journal of the Korea Institute of Information  
Security & Cryptology*, 29(6), 1339–1350.  
<https://doi.org/10.13089/JKIISC.2019.29.6.1339>

Appendix A

A.1 Project timeline

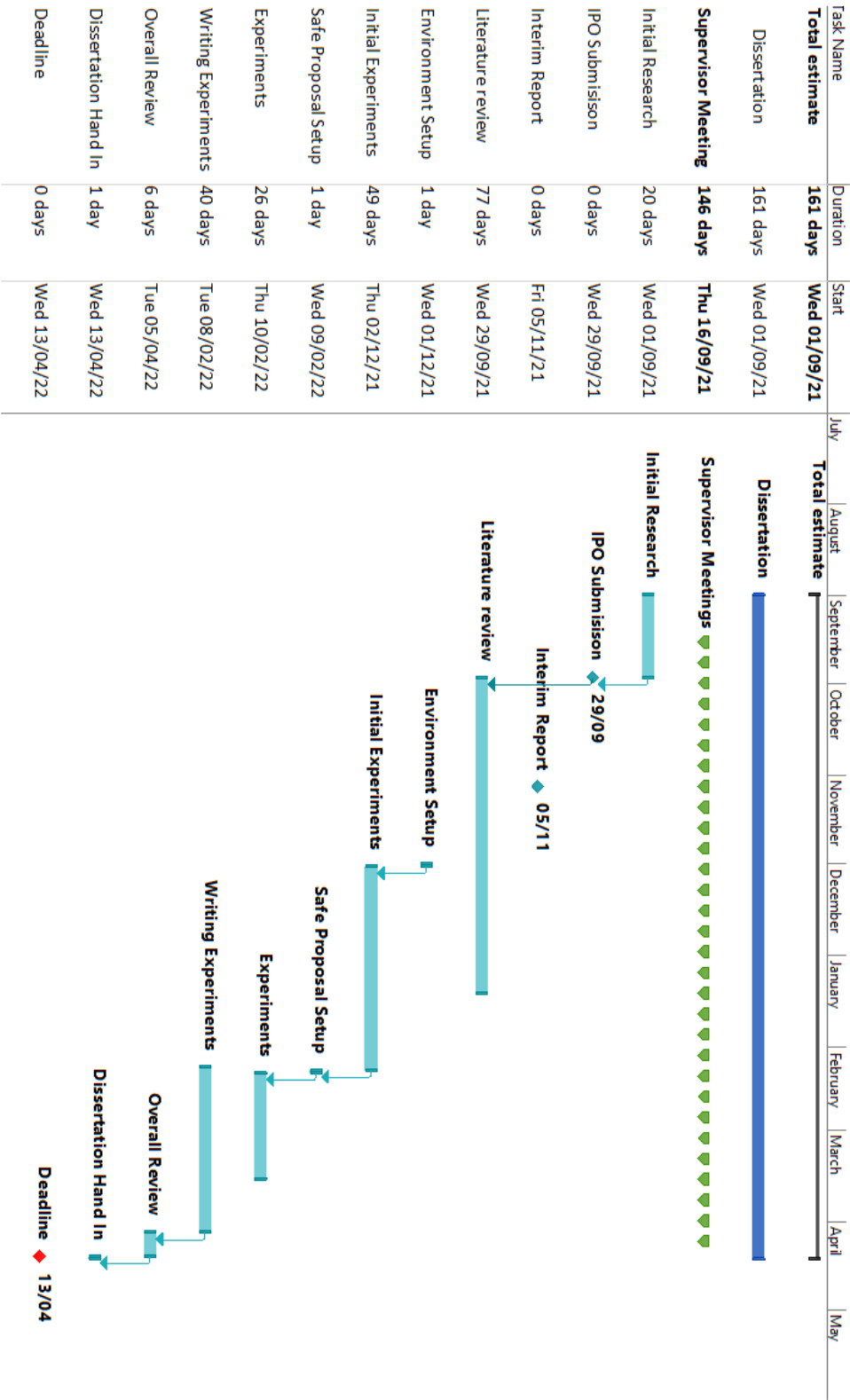


Figure 52 - Project Timeline

## A.2 Diary Examples

**EDINBURGH NAPIER UNIVERSITY**  
**SCHOOL OF COMPUTING**  
**PROJECT DIARY**


<b>Student:</b> Luis Loaysa	<b>Supervisor:</b> Macfarlane, Richard
<b>Date:</b> 15/09	<b>Last diary date:</b> N/A

**Objectives:**

Presented objectives and aims of the project to the supervisor and discuss possible routes to follow. Different interesting papers that could be used and on what we should focus more specifically for the literature review.

**Progress:**

First meeting.

 **Supervisor's Comments:**

Do further research on Docker containers and why this would be justified. Research more on ransomware and refine the aims of the project. For instance, check if there are ransomware attacks on Docker and why this is relevant.

☐

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

**Student:** Luis Loaysa

**Supervisor:** Macfarlane, Richard

**Date:** 02/12

**Last diary date:** 25/11

**Objectives:**

Advance literature review. Explain the importance of memory forensics in extracting keys from memory. Examples of this, important research done against ransomware, for example Davies paper and contrast it with current ones.

**Progress:**

Started using tools to recover keys from memory dumps. Using Docker in WLS (Windows Subsystem for Linux) with SSL to create AES keys that can be extracted with AES.

**Supervisor's Comments:**

Start developing a script that uses AES to encrypt files and contrast with Davies findings. Can his work be recreated?



EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

**Student:** Luis Loaysa

**Supervisor:** Macfarlane, Richard

**Date:** 20/01

**Last diary date:** 13/01

**Objectives:**

Using other tools against the memory dumps, RSAfinder, BulkExtractor, etc. Review literature review in case something else needs to be added. Think about the next experiments.

**Progress:**

AES keys were found in all the VMware memory dumps and files were decrypted using them. The script can now be paused for the key schedule to be in memory.

**Supervisor's Comments:**

Look at the literature of other ciphers such as Salsa20 and why it might be important. Try to build upon the research that already exists.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

**Student:** Luis Loaysa

**Supervisor:** Macfarlane, Richard

**Date:** 17/02

**Last diary date:** 10/02

**Objectives:**

Check if it is possible to use the script against a real sample of ransomware. If this is possible check how many keys can be recovered.

**Progress:**

The Salsa20 script it is able to identify and extract Salsa20 keys from the memory dumps after discovering the Initial State Matrix.

**Supervisor's Comments:**

We will need additional experiments to verify that this works in real ransomware and to see if we can decrypt files with it. If so, we will have many possibilities.

## Appendix B

### B.1 Encrypt\_AES

Encrypt\_AES is the Python script used in section 3.3.3 Initial Experiment Stage 1 to verify the method from the literature.

```
"""
@author: Luis Loaysa - 2022

Encrypts files with AES CBC mode.
"""

from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad
import time
import os

def encrypt(chunk):
    key = get_random_bytes(32)
    cipher = AES.new(key, AES.MODE_CBC)
    encrypted = cipher.encrypt(pad(chunk, AES.block_size))
    iv = cipher.iv
    ciphertext = iv + encrypted
    print('Key:', key.hex())
    print('IV:', iv.hex())
    return ciphertext

def main():
    file = input("Enter file name to encrypt: ")
    sleep_time = float(input("Enter in seconds the time to pause the
process: "))
    folder_path = os.getcwd()

    with open(folder_path + "\\ " + file, 'rb') as file:
        chunk = file.read()
        encrypted = encrypt(chunk)

    time.sleep(sleep_time)

    with open(folder_path + "\\ " + "data_enc.txt", 'wb') as enc:
        enc.write(encrypted)

if __name__ == '__main__':
    main()
```

## B.2 Decrypt\_AES

Decrypt\_AES is used in section 3.3.5 Initial Experiment Stage 3 to decrypt the file with the extracted AES key.

```
"""
@author: Luis Loaysa - 2022

Decrypt files encrypted with AES CBC mode.
"""

from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import os

def decrypt(chunk, key, IV):
    cipher = AES.new(key, AES.MODE_CBC, IV)
    decrypted = unpad(cipher.decrypt(chunk), AES.block_size)

    return decrypted

def main():
    file = input("Enter file to decrypt: ")
    key_hex = input("Enter key in hex: ")
    key = bytes.fromhex(key_hex)
    folder_path = os.getcwd()
    with open(folder_path + "\\\" + file, 'rb') as file:
        IV = file.read(16)
        chunk = file.read()
        decrypted = decrypt(chunk, key, IV)
        print("Decrypted text: ", decrypted)

    with open(folder_path + "\\\" + "data_recovered.txt", 'wb') as enc:
        enc.write(decrypted)

if __name__ == '__main__':
    main()
```

## B.3 Encrypt\_Salsa20

Encrypt\_Salsa20 was developed to identify Salsa20 key and nonce pairs in memory in section 3.4.3 and 3.3.7.3.

```
"""
@author: Luis Loaysa - 2022
```

```

Sodinokibi decryptor. Finds Salsa20 keys in memory and decrypts
files encrypted by Sodinokibi
"""

from Crypto.Cipher import Salsa20
from Crypto.Random import get_random_bytes
import os
import time

def salsa_encrypt(data):
    key = get_random_bytes(32)
    cipher = Salsa20.new(key=key)
    nonce = cipher.nonce
    encrypted = cipher.encrypt(data)
    print("Key: ", key.hex())
    print("Nonce: ", nonce.hex())
    return encrypted

def main():
    sleep_time = float(input("Enter in seconds the time to pause the
process: "))
    folder_path = os.getcwd()
    count = 0
    for file in os.listdir():
        count += 1
        print("Encrypting:", file)
        new_file = str(count) + "data_enc_Salsa20.txt"
        with open(folder_path + "\\\" + file, 'rb') as file:
            chunk = file.read()
            encrypted = salsa_encrypt(chunk)
            with open(folder_path + "\\\" + new_file, 'wb') as enc:
                enc.write(encrypted)
        time.sleep(sleep_time)

if __name__ == '__main__':
    main()

```

## B.4 Decrypt Salsa20

This script was used in sections 3.3.7.5 to decrypt files encrypted with Salsa20.

```

"""
@author: Luis Loaysa - 2022

Decrypts files encrypted with Salsa20.
"""

```

```
from Crypto.Cipher import Salsa20
import os

def decrypt(chunk, key, nonce):
    cipher = Salsa20.new(key=key, nonce=nonce)
    decrypted = cipher.decrypt(chunk)
    return decrypted

def main():
    file = input("Enter file to decrypt: ")
    key_hex = input("Enter key in hex: ")
    nonce_hex = input("Enter nonce in hex: ")
    key = bytes.fromhex(key_hex)
    nonce = bytes.fromhex(nonce_hex)
    folder_path = os.getcwd()
    with open(folder_path + "\\\" + file, 'rb') as file:
        chunk = file.read()
        decrypted = decrypt(chunk, key, nonce)
        print("Decrypted text: ", decrypted)
        print("Decrypted in hex:", decrypted.hex())

    with open(folder_path + "\\\" + "data_recovered.txt", 'wb') as enc:
        enc.write(decrypted)

if __name__ == '__main__':
    main()
```

## B.5 Null Bytes

Null Bytes script was used to corroborate that the encryption key and nonce pair extracted could decrypt the related files in section 3.4.5.1.

```
"""
@author: Luis Loaysa 2022

Decrypting Null Bytes of Sodinokibi
encrypted files
"""

from Crypto.Cipher import Salsa20

plaintext = '27D7F380'
plaintext_bytes = bytes.fromhex(plaintext)
key_hex =
'232371c7f5194c40f2c7696b4e48604439a8da6a32290044f4dd1b3a387e00f4'
secret = bytes.fromhex(key_hex)
nonce_hex = '3d61027bb8265ce8'
```

```
nonce = bytes.fromhex(nonce_hex)
cipher = Salsa20.new(key=secret, nonce=nonce)
decrypted = cipher.decrypt(plaintext_bytes)
print('Decrypting "27D7F380":', decrypted.hex())
```

## B.6 Shifting Bytes

The Shifting Bytes script in section 3.4.5.2 was developed to decrypt the files. Note that this script had the tested file in hexadecimal as variable “*my\_string*”, however for the sake of brevity it is omitted in this output.

```
"""
@author: Luis Loaysa 2022

Shifting bytes to find decryption position
"""

from Crypto.Cipher import Salsa20
import os

my_string = "A file is entered here as
             byte array"

begin = 0
end = 0

key_hex =
'edd8a9f8e446c41b312649df34f5237740db6b2a5923a5a44300389c9c3d3f54'
secret = bytes.fromhex(key_hex)

nonce_hex = '094f60cabbcfb4c9'
nonce = bytes.fromhex(nonce_hex)

match = 'ffd8ffe10018457869660000'
path_store = os.getcwd() + "\\\" + "testing"

for byte in range(int(len(my_string) / 2)):
    first_chunk = my_string[begin:]
    second_chunk = my_string[:end]
    begin += 2
    end += 2
    plaintext = first_chunk + second_chunk

    plaintext_bytes = bytes.fromhex(plaintext)

    cipher = Salsa20.new(key=secret, nonce=nonce)
    decrypted = cipher.decrypt(plaintext_bytes)
```

```
match = 'ffd8ffe10018457869660000'
if match in decrypted.hex():
    print('Success, offset:', byte, ' and value', plaintext[0:10])
    print(decrypted.hex()[0:70])
    print(decrypted.hex()[70:140])
    print(decrypted.hex()[140:210])
```

## B.7 PowerShell Automation

The PowerShell script was developed to automate the process of taking snapshots of the virtual machine at specified times.

```
cd "C:\Program Files (x86)\VMware\VMware Workstation"

$RunningVMs = .\vmrun list | select-object -skip 1
$counter = 10
start-sleep -seconds 4;

while($true)
{
    Foreach ($RunningVMs in $RunningVMs)
    {
        "Suspending $RunningVMs..."
        .\vmrun suspend "$RunningVMs"
    }

    $counter++

    .\vmrun snapshot "C:\Users\Experiment\Documents\Virtual
Machines\W10Sodinokibi\W10Sodinokibi.vmx" $counter

    .\vmrun start "C:\Users\Experiment\Documents\Virtual
Machines\W10Sodinokibi\W10Sodinokibi.vmx"

    start-sleep -seconds 1;
}
```

## B.8 Salsa20 Extraction Tool

The extraction tool has been used and updated throughout the experiments. In section 3.3.7.3 it identified and extracted Salsa20 key and nonce pairs from memory. However, in section 3.4.4 was improved to relate those pair to the files that had the respective nonces appended as metadata by the ransomware. After the decryption was successful, and as



part of the automation process, the decryption script was adapted and integrated to the Salsa20 extraction tool.

```

"""
@author: Luis Loaysa 2022

Sodinokibi decryptor. Finds Salsa20 keys in memory and decrypts
files encrypted by Sodinokibi
"""

import re
import os
from Crypto.Cipher import Salsa20

def decrypt_file(folder_src, file_key_nonce):
    """Decrypting found files with Salsa20 algorithm, nonce and key"""
    new_folder = os.getcwd() + "\\\" + 'Restored'
    for file in os.listdir(os.getcwd() + "\\\" + folder_src):
        for secret, file_nonce in file_key_nonce.items():
            if file == file_nonce[1]:
                rel_nonce = file_nonce[0] # Nonce is within list in a
dictionary
                with open(os.path.join(os.getcwd() + "\\\" +
folder_src, file), 'rb') as f:
                    chunk = f.read()
                    nonce = bytes.fromhex(rel_nonce) # converting to
bytes

                    secret = bytes.fromhex(secret)
                    size = len(chunk)
                    end_64 = chunk[-64:] # Start decrypting at the
end of the file -64 bytes
                    begin = chunk[:size - 64]
                    encrypted = end_64 + begin
                    cipher = Salsa20.new(key=secret, nonce=nonce)
                    decrypted = cipher.decrypt(encrypted)
                    if not os.path.exists(new_folder):
                        os.makedirs(new_folder)
                    file_bytes = bytes.fromhex(decrypted.hex()[128:-
336]) # Exact amount of metadata for the file
                    new_file = "recovered_" +
os.path.splitext(file)[0]
                    print('[*]', new_file)
                    fho = open(new_folder + "\\\" + new_file, 'wb')
                    fho.write(file_bytes)
                    fho.close()
                    f.close()

def search_file(incl_nonce, folder):

```

```

"""Finding files with a given nonce included in them"""
files_nonce = {}
for nonce in incl_nonce:
    for file in os.listdir(os.getcwd() + "\\\" + folder):
        with open(os.path.join(os.getcwd() + "\\\" + folder, file),
'rb') as f:
            reading = f.read().hex()[-232:]
            match = re.findall(nonce, reading)
            if match:
                files_nonce.update({file: nonce})

return files_nonce

def extracting_key(lists):
    """Matching and verifying keys found in memory"""
    filtered_keys = {}
    storing_keys = {}
    expand = [item for sublist in lists for item in sublist] #
Unpacking lists
    for key in expand:
        if key[40:48] == '6e642033' and key[120:128] == '7465206b':
            filtered_keys.update({key[8:40] + key[88:120]:
key[48:64]})
            # If second and fourth words exist in their location add
to dictionary
            # key[8:40] + key[88:120] -> Key key[48:64] -> Nonce
    for key, nonce in filtered_keys.items():
        if key not in storing_keys.values():
            storing_keys.update({key: nonce})

    return storing_keys

def reading_file(name_file):
    """Reading dump file to extract Salsa20 keys"""
    chunk_sz = 131072
    count = 0
    read_ahead = []
    storing_keys = []
    try:
        print('[+] Finding Salsa20 keys loaded in memory...', '\n')
        with open(name_file, 'rb') as f:
            while True:
                count += 1
                dump = f.read(chunk_sz)
                chunk = dump.hex()
                expand = re.findall(r'65787061.{128}', chunk)
                if expand:
                    storing_keys.append(expand)

```

```

        if count % 2 != 0:
            read_ahead.insert(0, dump[-500:])
        if count % 2 == 0:
            read_ahead.insert(0, dump[:500])

        gap_bytes = [b''.join(read_ahead[0:2])] # Join hex
bytes to avoid breaks from cutting bytes
        for gap in gap_bytes:
            gap_hex = gap.hex()
            expand = re.findall(r'65787061.{128}', gap_hex)
            if expand:
                storing_keys.append(expand)

        if len(read_ahead) == 20: # Prevents list getting too
big
            read_ahead = []
        if not dump:
            break
    except IOError:
        print('[+] Error opening the file')
        exit()

    return storing_keys

def main():
    """Main menu"""
    number_keys_found = 0
    name_file = input("[+] Enter the name of the memory dump and its
extension: ")
    key = reading_file(name_file)
    storing_keys = extracting_key(key)
    for key, nonce in storing_keys.items():
        number_keys_found += 1
        print('[*] 32 byte Salsa20 Key -', key)
        print('[*] Nonce -', nonce)
    print('[*] Total number of keys found:', number_keys_found)
    print("[+] That's all we could find!", '\n')
    with open("Sodinokibi_keys.txt", "w") as file_keys:
        for key, value in storing_keys.items():
            file_keys.write(key)
            file_keys.write(value)
            file_keys.write('\n')

    incl_nonce = storing_keys.values()
    if len(storing_keys) >= 1:
        answer_files = input("[+] Find files for these keys and
decrypt them? Y or N: ").lower()
        if answer_files == 'y':
            file_key_nonce = {}
            folder_src = input("[+] Enter folder with files: ")

```

```

files_nonce = search_file(incl_nonce, folder_src)
for f, n in files_nonce.items():
    print("[*] Nonce:", n, 'in file:', f, 'found.')
# files_nonce contain file as key and nonce as value
for file, nonce in files_nonce.items():
    list_file_nonce = []
    if nonce in storing_keys.values():
        keys = list(storing_keys.keys()) # Keys converted
to list
        nonce_list = list(storing_keys.values()) # Values
converted to list
        nonce_position = nonce_list.index(nonce) # Select
nonce from list
        key_extract = keys[nonce_position] # Store key
for the specific nonce
        list_file_nonce.append(nonce)
        list_file_nonce.append(file)
        file_key_nonce[key_extract] = list_file_nonce #
Create dictionary with file, key and noce
    if len(files_nonce) >= 1:
        print('[+] Decrypting files... placing them in
"Restored_files"')
        decrypt_file(folder_src, file_key_nonce)
    else:
        exit()
else:
    exit()

print("[+] That's all we could decrypt!")

if __name__ == '__main__':
    main()

```