

Employing Encryption Workarounds to Support Digital Forensics Investigations

Dionisios Aristotelis Kalochristianakis

Thesis submitted in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *Evangelos Markatos*

Thesis Co-Advisor: Dr. *Harry Manifavas*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS) and the European Union's Internal Security Fund - Police (ISFP) programme under grant agreement No 101038738.

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

**Employing Encryption Workarounds to Support Digital Forensics
Investigations**


Thesis submitted by
Dionisios Aristotelis Kalochristianakis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science


THESIS APPROVAL

Author:


Dionisios Aristotelis Kalochristianakis

Committee approvals:

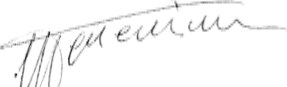

Evangelos Markatos
Professor, Thesis Supervisor


Harry Manifavas
PhD, Thesis Co-Advisor


Yannis Tzitzikas
Professor, Committee Member


Kostas Magoutis
Associate Professor, Committee Member

Departmental approval:


Polyvios Pratikakis
Associate Professor, Director of Graduate Studies

Heraklion, February 29, 2024

Employing Encryption Workarounds to Support Digital Forensics Investigations

Abstract

The use of encryption in our everyday lives is ubiquitous. For example, encryption is utilized to secure communications between two parties, authenticate users when logging to a service or even certify that websites are legitimate. However, criminals can also benefit from the use of encryption hiding their traces of illicit activities and making law enforcement agencies unable to proceed with their criminal investigations due to being locked out of digital evidence. During the last decade the problem became more apparent as more and more law enforcement agencies reported being blocked from investigating criminals due to the abuse of encryption by them.

This sparked the “Going Dark” Debate across the globe. From one hand, law enforcement and intelligence agencies pushed for weakened encryption policies and establishing encryption countermeasures for lawful access. On the other hand, institutions for human rights condemned this approach because it would diminish the privacy and security of everyone. With no clear solution to the debate on policy level, the concept of encryption workarounds was introduced that bridges the gap between the legal and the technological aspects of the debate.

To support this approach, we present existing encryption workarounds that are able to bypass encryption or leverage weaknesses in the cryptographic implementation of certain desktop software applications, with a focus on Windows systems. More importantly we categorize the methods into filesystem and memory analysis, we test and modify them to make them function properly. Additionally we validate that they meet the criteria of being forensically sound so that they can be used in criminal investigations by law enforcement and be accepted in a court of law.

Finally, we present AWLPS, a platform that performs several encryption workarounds through a Graphical User Interface. AWLPS, is designed to provide data integrity and a forensically sound methodology when decrypting electronic evidence. Furthermore, it can be expanded through a Plugin architecture, allowing for the integration of new encryption workaround modules, thus enhancing the platform’s durability over time.

Εκμετάλλευση Παρακάμψεων Κρυπτογραφίας για την Υποστήριξη Ψηφιακών Εγκληματολογικών Ερευνών

Περίληψη

Η χρήση της κρυπτογράφησης στην καθημερινή μας ζωή είναι πανταχού παρούσα. Για παράδειγμα, η κρυπτογράφηση χρησιμοποιείται για την ασφάλεια των επικοινωνιών μεταξύ δύο μερών, την ταυτοποίηση των χρηστών κατά τη σύνδεσή τους σε μια υπηρεσία ή ακόμα και την επιβεβαίωση της εγκυρότητας ιστότοπων. Ωστόσο, οι εγκληματίες μπορούν επίσης να επωφεληθούν από τη χρήση κρυπτογράφησης για να κρύψουν τα ίχνη των παράνομων δραστηριοτήτων τους και να εμποδίζουν τις αστυνομικές αρχές να προχωρήσουν στις ποινικές τους έρευνες εξαιτίας του ότι αποκλείονται από τα ψηφιακά πειστήρια.

Την τελευταία δεκαετία το πρόβλημα έγινε πιο εμφανές καθώς ολοένα και περισσότερες αστυνομικές αρχές ανέφεραν ότι εμποδίζονταν από τις έρευνες εγκληματιών λόγω της κατάχρησης της κρυπτογράφησης από αυτούς. Αυτό ξεκίνησε τη συζήτηση 'Going Dark' σε ολόκληρο τον κόσμο. Από τη μία πλευρά, οι αστυνομικές αρχές και οι υπηρεσίες πληροφοριών προώθησαν πολιτικές αποδυνάμωσης της κρυπτογράφησης και την εγκατάσταση αντιμέτρων κρυπτογράφησης με σκοπό να μπορούν να έχουν νόμιμη πρόσβαση στα δεδομένα των υπόπτων. Από την άλλη πλευρά, μη κυβερνητικοί οργανισμοί προάσπισης των ανθρωπίνων δικαιωμάτων καταδίκασαν αυτήν την προσέγγιση διότι θα μείωνε την ιδιωτικότητα και την ασφάλεια όλων. Χωρίς καμία σαφή λύση στη συζήτηση σε πολιτικό και νομικό επίπεδο, εισήχθη ο όρος 'παρακάμψεις κρυπτογράφησης' που γεφυρώνει το χάσμα μεταξύ των νομικών και τεχνολογικών πτυχών της συζήτησης.

Για να υποστηρίξουμε αυτήν την προσέγγιση, παρουσιάζουμε υπάρχουσες παρακάμψεις κρυπτογράφησης που είναι σε θέση να παρακάμψουν την κρυπτογράφηση ή να εκμεταλλευτούν αδυναμίες στην κρυπτογραφική εφαρμογή ορισμένων εφαρμογών λογισμικού για συσκευές δεσχοπ, με έμφαση στο λειτουργικό Windows. Κατηγοριοποιούμε τις μεθόδους σε ανάλυση filesystem και μνήμης, τις δοκιμάζουμε και τις τροποποιούμε για να λειτουργούν ορθά. Επιπλέον, επικυρώνουμε ότι πληρούν τα κριτήρια της διαδικασίας διατήρησης πειστηρίων, ώστε να μπορούν να χρησιμοποιηθούν σε εγκληματολογικές έρευνες από τις αστυνομικές αρχές και να γίνονται αποδεκτές σε δικαστήρια.

Τέλος, παρουσιάζουμε το AWLPS, μια πλατφόρμα που εκτελεί διάφορες παρακάμψεις κρυπτογράφησης μέσω γραφικού περιβάλλοντος χρήστη. Το AWLPS σχεδιάστηκε για να παρέχει ακεραιότητα δεδομένων και μια μεθοδολογία διατήρησης πειστηρίων όταν αποκρυπτογραφεί ψηφιακά πειστήρια. Επιπλέον, μπορεί να επεκταθεί μέσω αρχιτεκτονικής Plugin, επιτρέποντας την ενσωμάτωση νέων μεθόδων παρακάμψεων κρυπτογράφησης, ενισχύοντας έτσι την ανθεκτικότητα της πλατφόρμας με την πάροδο του χρόνου.

Acknowledgements

I would like to express my sincere gratitude to my thesis advisors, Evangelos Markatos & Harry Manifavas, for their unwavering support, invaluable guidance, and insightful feedback throughout the entire process of researching and writing this thesis. Their expertise, encouragement, and patience have been instrumental in shaping the direction of this work.

I am also deeply thankful to my friends and colleagues: Gianni, Kosta, Mano, Kosta, Skerdi & Kosta for their valuable contributions, encouragement, and feedback during various stages of this research. Their expertise and constructive criticism have greatly enriched the quality of this thesis.

I am grateful to the Institute of Computer Science and the CYBERSPACE project for providing the necessary resources and facilities for conducting this research. Their support has been essential in facilitating the smooth progression of this study.

Last but not least, special thanks are due to my family for their unwavering love, encouragement, and belief in my abilities. Their constant support and understanding have been my pillars of strength throughout this journey.

στους γονείς μου

Contents

Table of Contents	i
List of Tables	iii
List of Figures	v
1 Introduction	1
1.1 Areas of interest	1
1.2 The “Going Dark” debate	3
1.3 Thesis Contributions	4
1.4 Outline	5
2 Background	7
2.1 Cryptography	7
2.1.1 Symmetric Encryption	7
2.1.2 Asymmetric Encryption	8
2.1.3 Hashes	9
2.2 Digital Forensics	9
2.2.1 Principles	10
2.2.2 The digital forensics investigation process	11
2.2.3 Legal principles	12
2.3 Criminal Use of Encryption & Countermeasures	13
2.3.1 The encryption debate	13
2.3.2 Encryption workarounds	15
3 Methodology	19
3.1 File system analysis	20
3.1.1 Mozilla software	20
3.1.1.1 Architecture	21
3.1.1.2 Scenario	23
3.1.1.3 Setup	23
3.1.1.4 Validation	29
3.1.2 Chrome Decrypt	31
3.1.2.1 Architecture	32

3.1.2.2	Scenario	34
3.1.2.3	Setup	35
3.1.2.4	Validation	37
3.1.3	Meo encryption software	40
3.1.3.1	Architecture	40
3.1.3.2	Scenario	42
3.1.3.3	Setup	43
3.1.3.4	Validation	43
3.2	Memory analysis	44
3.2.1	BitLocker	44
3.2.1.1	Architecture	44
3.2.1.2	Scenario	48
3.2.1.3	Setup	48
3.2.1.4	Validation	52
3.2.2	KeePass	53
3.2.2.1	Architecture	53
3.2.2.2	Scenario	53
3.2.2.3	Setup	54
3.2.2.4	Validation	58
4	Implementation	59
4.1	Related Tools & Principles	59
4.1.1	Extensibility	60
4.1.2	Forensic Soundness	62
4.1.3	Operating System Interoperability	62
4.1.4	Shortfalls with regards to our Encryption	63
4.2	Architecture	64
4.2.1	Plugin Architecture	65
4.2.2	User Interface	68
4.2.2.1	AWLPS Analysis of Windows Artifacts	69
4.2.2.2	AWLPS Memory Analysis	72
4.2.3	Volatility 3 Integration	75
5	Conclusion	77
5.1	Future work	77
5.1.1	Forensic Image Loading & Handling	77
5.1.2	Additional Workarounds	77
5.1.3	Reporting & Documentation	78
5.1.4	User Testing	78
5.2	Ethics	79
5.3	Summary	79

A	Appendix A	81
A.1	Taxonomy of encryption workarounds included in AWLPS	81
A.2	Forensic Drive Acquisition using Guymager	83
B	AWLPS Installation and Extension	85
B.1	AWLPS Installation & Configuration	85
B.1.1	Linux	85
B.1.2	Windows	86
B.2	AWLPS Plugin Extension	88
B.2.1	Scenario	88
B.2.2	Implementation	88
	Bibliography	93

List of Tables

3.1	MD5 & SHA256 hashes of “logins.json” before and after running the decryption workaround	29
3.2	MD5 & SHA256 hashes of “key4.db” before and after running the decryption workaround	30
3.3	List of fields and their respective types of the logins table that Chrome uses to store passwords	32
3.4	MD5 & SHA256 hashes of the files associated with the DPAPICK method for Google Chrome.	38
3.5	MD5 & SHA256 hashes of the “Local State” file.	38
3.6	MD5 & SHA256 hashes of the masterkey file.	39
3.7	Process of user password storage	43
3.8	MD5 & SHA256 hashes of the memory dump	52
3.9	MD5 & SHA256 hashes of the process & memory dump files before and after running the exploit	58
4.1	Categories of post-exploitation modules in metasploit framework.	61
4.2	Module fields in AWLPS configuration file.	67

List of Figures

1.1	Type of problems EU member states face when dealing with encryption.	3
1.2	Number of locked devices from cases of the Manhattan DA's office.	4
2.1	Symmetric key encryption and decryption process.	8
2.2	Asymmetric key encryption and decryption process.	9
2.3	Hashing process.	10
2.4	The process of a digital forensics investigation.	11
2.5	Kerr & Schneier's encryption workarounds taxonomy.	17
3.1	Mozilla Firefox prompt to save web credentials when logging in a website.	20
3.2	Decryption process of saved user web credentials in Mozilla software such as Firefox.	22
3.3	How to navigate to the Firefox saved logins and passwords page.	24
3.4	Firefox Passwords page.	25
3.5	Firefox decrypted passwords.	26
3.6	Thunderbird decrypted passwords.	27
3.7	Open TOR Browser settings.	28
3.8	Settings that need to be enabled to allow TOR to store passwords.	28
3.9	TOR decrypted passwords.	29
3.10	Google Chrome prompt to save credentials	31
3.11	Password encryption process in Chrome	33
3.12	High-level overview of Meo encryption process.	40
3.13	Process of user password storage.	41
3.14	Header of an encrypted file with Meo.	41
3.15	Meo email encryption.	42
3.16	Received email encrypted by Meo.	42
3.17	BitLocker decryption process	45
3.18	Example kernel pool allocation [41].	47
3.19	Live memory acquisition with FTK Imager	49
3.20	Memory dump information extracted by the Volatility framework.	50
3.21	List partitions of the Windows drive.	51
3.22	Unlocking BitLocker drive and mounting it at <code>/mnt/drive</code>	51

3.23	Contents of the unlocked BitLocker drive.	52
3.24	Dump process memory from Windows task manager	56
3.25	Output of the PoC using the memory dump	57
3.26	Output of the PoC using the process dump	57
4.1	AWLPS architecture diagram.	64
4.2	Dialogue selection for dictionaries inside the AWLPS tool.	68
4.3	Menu Tree of the AWLPS tool.	69
4.4	AWLPS landing page.	70
4.5	User input dialog in AWLPS to load an EWF file.	71
4.6	AWLPS analysis of Windows encrypted artifacts.	72
4.7	AWLPS instance of a failed module displayed in red.	73
4.8	Details of the mozilla decryption module in AWLPS.	73
4.9	AWLPS memory analysis screen.	74
4.10	BitLocker FVEK extraction from acquired memory using Volatility 3.	75
4.11	Extraction of BitLocker FVEK from the AWLPS.	76
A.1	List of all the modules offered in AWLPS based on each category introduced by Kerr & Schneier [39].	82
A.2	Guymager - drive selection	84
A.3	Guymager - acquisition process	84

Chapter 1

Introduction

In today's tech-driven landscape, the role of electronic devices in criminal investigations is crucial [28, 31]. Even the EU Commission states that 85% of criminal investigations require electronic evidence to proceed to courts[20]. However, the widespread use of encryption in popular software and devices, intended to protect user privacy, presents a challenge for law enforcement. While encryption is generally seen as a positive for citizens' privacy, it's also exploited by some bad actors looking to avoid prosecution.

Encryption is now a standard feature, with many mainstream messaging apps offering end-to-end encryption. Despite its benefits, end-to-end encryption (E2E) comes with some serious downsides.

1.1 Areas of interest

Cybercriminals are not the only group that abuses the use of encryption in ransomware and to secure their communications in order to hide their illicit activities. The widespread adoption of end-to-end encryption by mainstream communication platforms has also resulted in a significant surge in its use among regular criminals lacking advanced technical expertise.

Europol reported in 2016 that 13 member states observed criminals employing encryption software to encrypt their data [12], hindering criminal investigations, while 8 member states identified encryption as a major challenge in combating cybercrime. A year later, Europol noted that 20 member states reported encountering encryption almost invariably in criminal investigations [13], particularly in areas like dark web marketplaces, child sexual exploitation material, and terrorism.

1. The Dark Web and online illegal marketplaces

Cybercriminals are the most obvious category of criminals that abuse encryption to hide their illicit acts. They possess the knowledge to assess the

security of an application and to remain hidden using TOR¹ and bulletproof hosting². However, it is important here to note that in most cases they too tend to use common end to end encrypted applications such as Jabber³ and Internet Relay Chat (IRC) [36] or ICQ⁴ to a lesser extent [12].

As far as illegal marketplaces are concerned, a lot of them migrated from the dark web to end to end encrypted messaging applications that serve users under the protection of end to end encryption. This has led to new initiatives being developed such as the **Televend**[16, 54] vending service bot, which benefited from Telegram’s end-to-end encryption in order to sell illegal substances.

2. Online Child Sexual Exploitation Material (CSEM)

In cases involving CSEM it has been observed by LEAs [14, 15, 16] that criminals have shifted from using peer to peer networks hosted in the dark web to distribute CSEM to common messaging end to end encrypted applications. Such applications used for these purposes are WhatsApp⁵, Viber⁶ and Telegram⁷.

In 2017 an international investigation dubbed “Operation Tantalio” led by Interpol managed to dismantle a large CSEM network. The operation was launched in 2016 by the Spanish National Police with a focus on the TOR network. The investigators managed to find links to invite-only WhatsApp groups that shared explicit CSEM [34].

3. Terrorism

Terrorism is another major example where criminal abuse of the encryption is observed. In [14], Europol mentions that terrorist groups do not tend to launch cyber attacks. Rather they tend to communicate using encrypted messaging apps and even moved their recruitment platforms from Facebook and Twitter to **end-to-end encrypted** messaging platforms such as Threema⁸, Signal⁹ and Telegram. They have also developed an Operational Security Manual that advises their members on how to stay secure on the internet [72].

¹Tor Project, www.torproject.org/

²Sentinel One: What is Bulletproof Hosting, www.sentinelone.com/cybersecurity-101/bulletproof-hosting/

³Cisco Jabber, www.cisco.com/c/en_in/products/unified-communications/jabber/index.html

⁴ICQ, icq.com/desktop

⁵WhatsApp, www.whatsapp.com/

⁶Viber, www.viber.com/en/download/

⁷Telegram, telegram.org/

⁸Threema, threema.ch/en

⁹Signal, signal.org/download/

1.2 The “Going Dark” debate

In 2015, the FBI obtained the iPhone of a terrorist that was responsible for an attack that killed 14 people in San Bernardino, California. The FBI unsuccessfully attempted to access his iPhone as part of their investigation to find out if there were more accomplices. The Department of Justice obtained a court order to make Apple comply and unlock the phone. Instead, Apple refused and then started a public trial between Apple and the Department of Justice that sparked various LEAs saying that they were “Going Dark” in investigating criminals due to the misuse encryption. Eventually, the FBI bought an exploit chain from a private cybersecurity firm and was able to bypass the iPhone encryption and get access to the data pertinent to the case [18].

In 2016, the Council of the European Union conducted a questionnaire among EU member states to assess how European Law Enforcement Agencies (LEAs) handle encryption in criminal investigations [19]. Subsequently declassified and made public, the responses from member states to the question “What main issues do you typically encounter when seizing encrypted evidence and decrypting it?” are illustrated in Figure 1.1. Among the 18 member states that participated in the questionnaire, all reported facing technical issues when decrypting seized data.

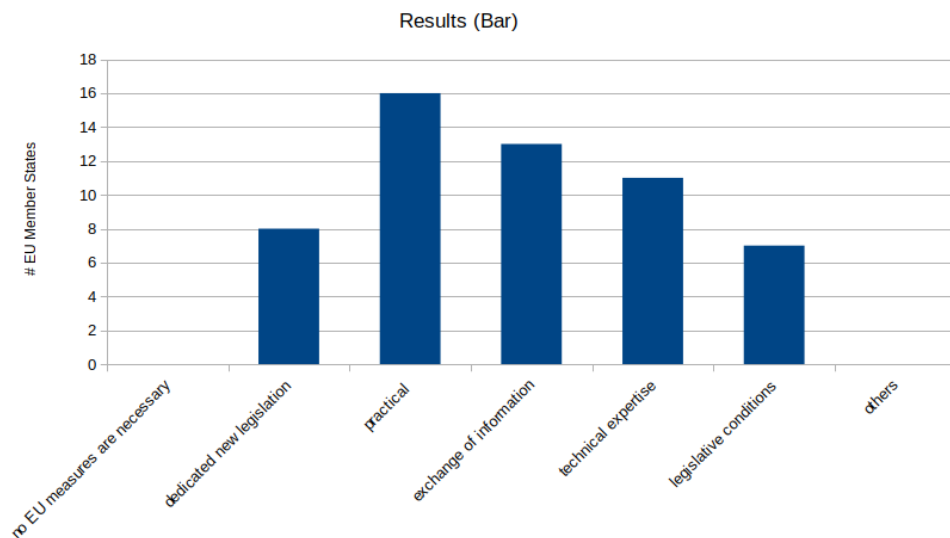


Figure 1.1: Type of problems EU member states face when dealing with encryption.

In 2019 the Manhattan District Attorney’s Office published the report “Smartphone encryption and public safety” [52] smartphone devices were essential in the prosecution of criminal cases even outside of the scope of the cyberspace. Figure 1.2 from the same report shows that the quantity of encrypted devices that the DA receives as evidence increases every year.

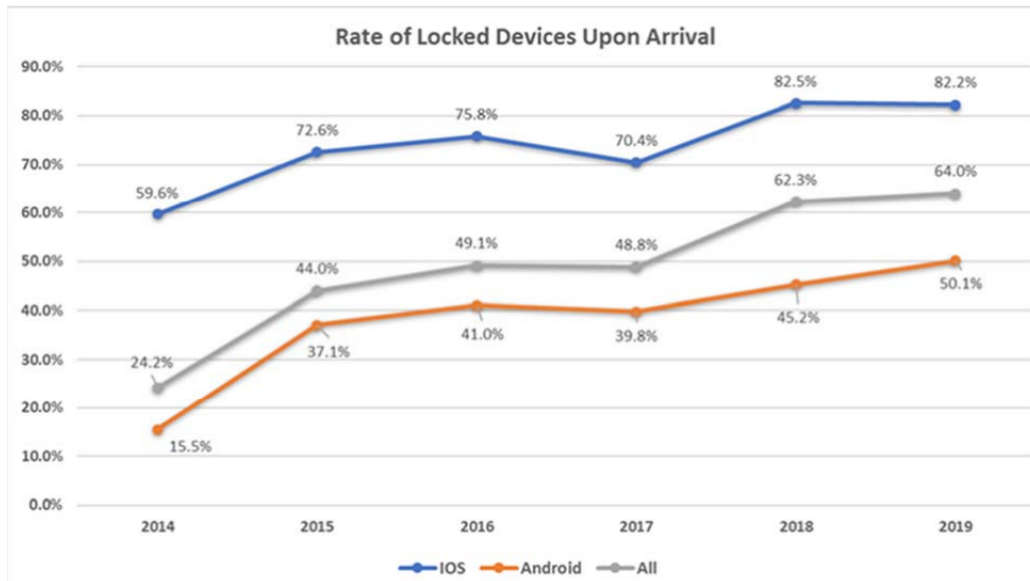


Figure 1.2: Number of locked devices from cases of the Manhattan DA’s office.

In 2021, the Signal Foundation owner of the Signal application for mobile and desktop devices, published a subpoena they received from the United States Attorney’s Office in the Central District of California. The subpoena ordered Signal to disclose to law enforcement personal data such as message history, location and media shared for a specific suspect. Signal responded to the subpoena that it was unable to comply with the request since the only thing that they collect from the users and is not end-to-end encrypted was UNIX timestamps¹⁰ from when he created the account and his last login date [65].

While many countries have established legal framework permitting LEAs to lawfully access a suspect’s device for crime investigation, the technical capabilities of LEAs in this regard often fall behind. The challenge is exacerbated by the growing prevalence of encryption in both devices and software. This not only ensures that the issue persists but also contributes to its escalation over time.

1.3 Thesis Contributions

Our contribution in this work includes:

1. An analysis of existing encryption workarounds targeting the following desktop software:
 - Filesystem analysis

¹⁰What is unix epoch: www.epoch101.com/

- Meo software encryption
 - Mozilla Software (Firefox, Thunderbird, TOR)
 - Google Chrome, Microsoft Edge, Opera browser
 - Memory analysis
 - Keepass password manager
 - BitLocker
2. A categorization of each encryption workaround based on the encryption workarounds taxonomy presented by Schneier et al. [39].
 3. An evaluation of the above-mentioned encryption workarounds and a validation of their forensic soundness.
 4. An implementation of a Volatility 3 plugin to extract the Full Volume Encryption Key of BitLocker out of a memory sample.
 5. A review between state-of-the-art open source security tools, their features that benefit our case and their limitations.
 6. Our implementation, AWLPS: an addition to the ALPS tool introduced in [3]. It is written in Python and provides automation in performing the aforementioned encryption workarounds while also providing a user friendly GUI and a module base for future extensions.

1.4 Outline

The structure of the remaining sections in the thesis is outlined as follows:

Chapter 2 delves into the fundamental concepts of cryptography, digital forensics, and addresses the challenges associated with encryption misuse, presenting proposed solutions.

In Chapter 3, a comprehensive explanation of each encryption workaround module is provided, along with a detailed account of their validation for forensic soundness. The intricacies of the methodology are explored in this section.

Moving on to Chapter 4, first we perform a review on existing state-of-the-art open source security tools listing their features and their limitations with regards to encryption. Then, attention is directed towards the presentation of our tool, AWLPS which implements the above mentioned features and addresses the previous limitations. This chapter presents the automated functioning of AWLPS in executing encryption workarounds. Moreover, we discuss our own implementation of a Volatility 3 plugin to extract BitLocker keys from memory dumps.

Finally, Chapter 5 wraps up by suggesting potential future extensions in the research of encryption workarounds and enhancements to our tool. Ethical considerations surrounding encryption workarounds and the research methodology are also addressed in the final chapter.

Chapter 2

Background

Within this chapter, we present a comprehensive exploration of topics essential to our work. To begin, we offer an overview of cryptography, categorizing it into three common types. Following that, we delve into the realm of digital forensics; its principles, challenges, and the process of a digital forensics investigation. Lastly, we examine the misuse of encryption, outlining our literature review, which draws from articles and reports sourced from law enforcement agencies, shedding light on how encryption hinders criminal investigations.

2.1 Cryptography

Cryptography, a cornerstone in the realm of information security, is both an “art and a science that employs mathematical algorithms, principles, means, and methods for the transformation of data in order to hide their semantic content, prevent their unauthorized use, or prevent their undetected modification”¹.

In a modern context, cryptography serves as a subsystem within a larger information or computer system, fortifying its defenses by maintaining the confidentiality of data and ensuring their integrity and authenticity. The ubiquitous presence of cryptography in modern computing plays a pivotal role in safeguarding online transactions, protecting sensitive information, and forming the foundation of secure communication protocols. This, in turn, contributes significantly to upholding the integrity and privacy of digital interactions.

2.1.1 Symmetric Encryption

Symmetric encryption relies on a single key to perform both the encryption and decryption processes. Usually the decryption process is the exact same as the encryption process. Figure 2.1 illustrates the process of encrypting a plaintext and transforming it to an unreadable ciphertext through a symmetric key algorithm and a symmetric encryption key. Then, the same encryption key is used along

¹See NIST Glossary on “cryptography”, csrc.nist.gov/glossary/term/cryptography

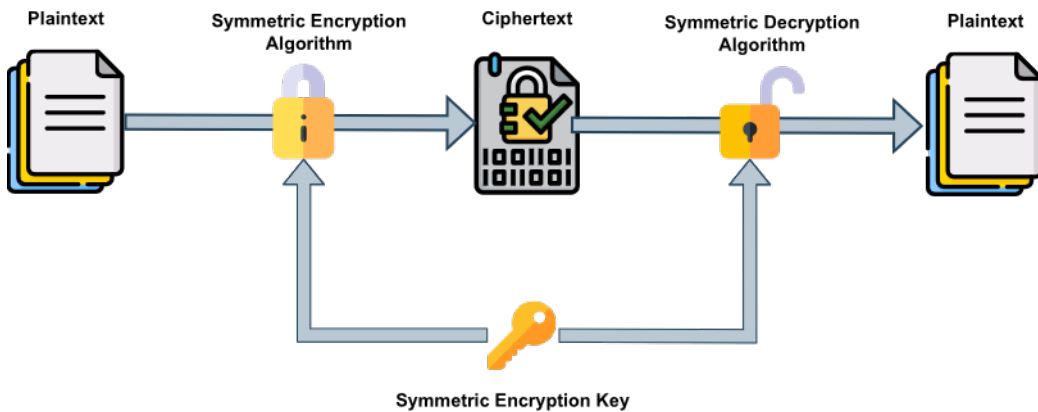


Figure 2.1: Symmetric key encryption and decryption process.

with the symmetric decryption algorithm to transform the ciphertext to the original plaintext. Generally, symmetric encryption is used to encrypt data *at-rest* such as passwords in passwords managers, text messages from messaging applications stored locally in a device or even in full device encryption. Some common symmetric encryption algorithms are OTP and AES [56] which can have multiple variations.

2.1.2 Asymmetric Encryption

Asymmetric encryption operates similarly with the symmetric encryption with one big difference. Instead of having one key to do both the encryption and decryption, there are two different keys used for encryption process. The keys are generated together as a key pair, either key can be used to encrypt and only the counterpart can be used to decrypt the ciphertext back to the plaintext. Asymmetric encryption is mainly used to securely establish communication with HTTPS through TLS.

Figure 2.2 illustrates the process of encryption and decryption using an asymmetric encryption algorithm. First, the plaintext is transformed to an unreadable ciphertext using an asymmetric encryption key and an asymmetric encryption algorithm. Then, the same ciphertext can be transformed back to the original plaintext by using the corresponding asymmetric decryption algorithm and the counterpart of the key pair. Some notable asymmetric encryption algorithms are RSA[60], DSA[58], DH[57] which can have multiple variations, and the more recent post quantum algorithms proposed in the NIST PQC project².

²NIST Post-Quantum Cryptography csrc.nist.gov/projects/post-quantum-cryptography

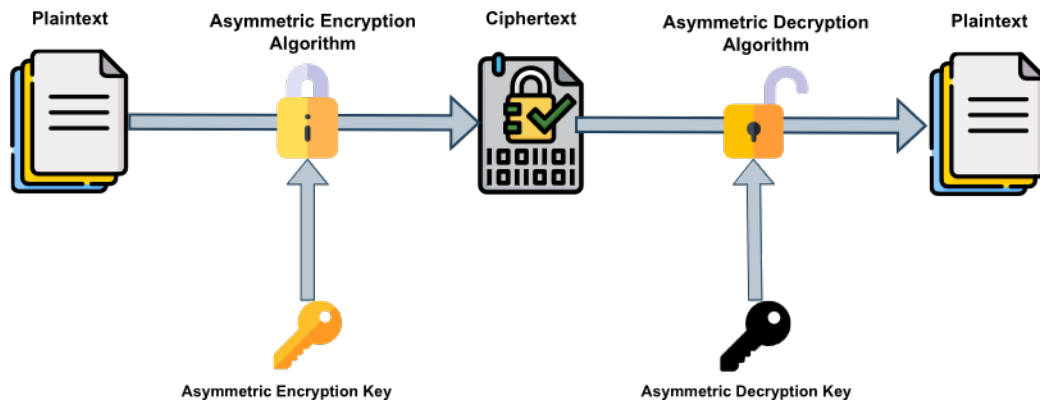


Figure 2.2: Asymmetric key encryption and decryption process.

2.1.3 Hashes

Hashes are one-way mathematical functions that transform any given input size into a fixed length output. The output, hash value is often times used as a signature to fingerprint and identify files, verify that the data remained unchanged or even securely store user passwords. Apart from the irreversibility of the hash value back to the original input, a good hash function should also have “**collision resistance**”. This property states that it is computationally infeasible to find two different inputs that produce the same output. This feature drastically limits the likelihood that two distinct inputs will share the exact same hash value.

Another important feature is the “**avalanche**” effect. This causes a slight change in the hash input to have a drastic change in the produced output. This is very important especially when storing hashed passwords. Even if the hashed passwords eventually leak, an attacker would not be able to guess the correct input of the hash function i.e. the password, by comparing how close the hash output is to some other already known values.

Some notable hash functions are MD5 [59], SHA-1 [61], SHA-2 [62] and SHA-3 [63].

2.2 Digital Forensics

Digital forensics, is a branch of forensic science that involves the “application of computer science and investigative procedures involving the examination of digital evidence - following proper search authority, chain of custody, validation with mathematics, use of validated tools, repeatability, reporting, and possibly expert testimony”³.

As defined in the first Digital Forensics Research Workshop [8]. “The use of

³This is the first definition of “Digital Forensics” by NIST (CNSSI 4009-2015 from DoDD 5505.13E) available here: csrc.nist.gov/glossary/term/digital_forensics

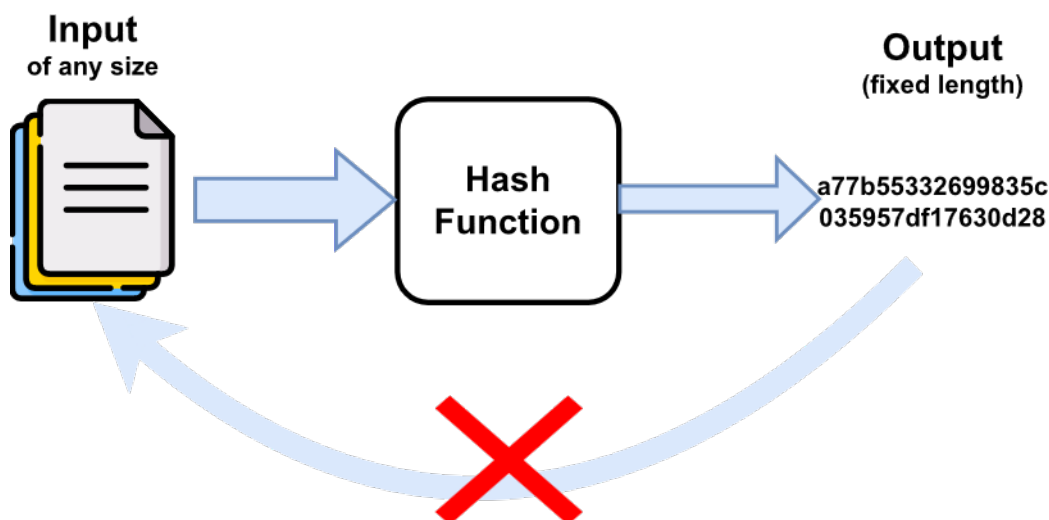


Figure 2.3: Hashing process.

scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations”.

2.2.1 Principles

In the book titled “Digital Forensics,” Arnes et al. [73] outline the fundamental principles of digital forensics, as described below.

- **Evidence integrity**

“Evidence integrity” refers to the preservation of evidence in its original form. This is a requirement that is equally valid both for the original evidence when it is collected, as well as for the copy of the evidence that is used for analysis and then referred to when evidence is presented to a court.

- **Chain of custody**

“Chain of custody” refers to the tracking of evidence through its collection, safeguarding, and analysis lifecycle by documenting each person who handled the evidence, the date/time it was collected or transferred, and the purpose for the transfer ⁴.

- **Forensic soundness**

An investigation is “forensically sound” if it adheres to established digital forensics principles, standards, and processes.

⁴See NIST glossary, csrc.nist.gov/glossary/term/chain_of_custody

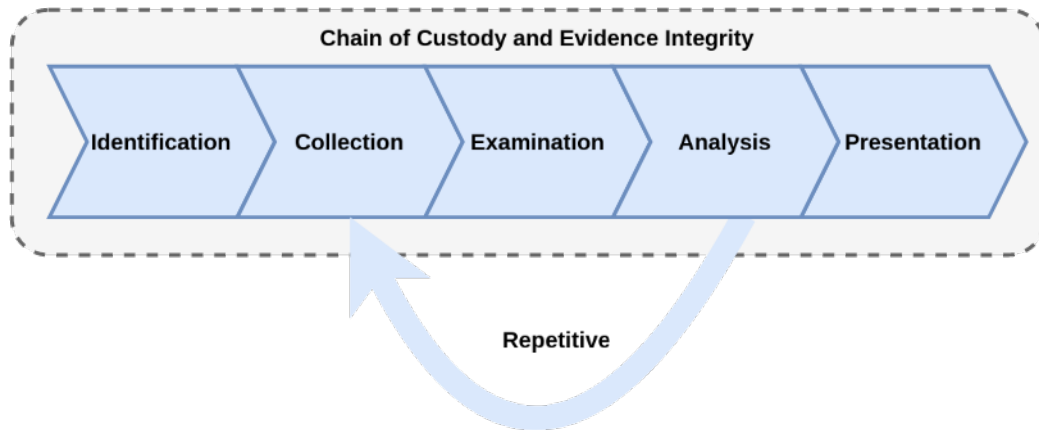


Figure 2.4: The process of a digital forensics investigation.

2.2.2 The digital forensics investigation process

Digital evidence can be discovered in a variety of sources, including computers, mobile devices, internet infrastructure, industrial systems, and other digital devices. The application of the forensic process and its fundamental principles is essential to ensure the integrity and reliability of an investigation. In this section, we outline the five phases of the digital forensics investigation process as presented in the book “Digital Forensics” [73], based on the principles of digital forensics and informed by established practices in both law enforcement and industry.

- **Identification**

The identification phase forms the basis for all the following phases of the investigation process. In this phase it is important to identify which digital objects (computers, phones, car infotainment systems etc.) are useful as evidence during the investigation.

An investigation may concentrate on several objectives, including identifying supporting information to substantiate a case, pinpointing information that refutes a hypothesis for a case, or validating the authenticity and accuracy of provided information.

Additionally, the investigator needs to be adequately prepared to investigate a case. In this phase, it is crucial to ensure the proper functioning of all forensic tools in their arsenal and verify the forensic soundness of the methods to be employed in subsequent phases.

- **Collection**

In the collection phase, the investigator meticulously duplicates data from all confiscated devices, taking utmost care to avoid any alterations. This precaution is essential because forensic analysis is ideally conducted on forensic

copies to preserve the original data source. Typically, the forensic technician overseeing evidence acquisition utilizes a combination of software and hardware write blockers. These tools ensure that the digital device is parsed in read-only mode, preventing any unintentional modifications to the data.

- **Examination**

Examining data often involves reorganizing, parsing, and preprocessing raw data to ensure it is comprehensible for forensic investigators during subsequent analysis. To streamline this process, analysts commonly employ forensic tools and techniques tailored for extracting information relevant to the case.

- **Analysis**

Following the examination phase, data undergoes preparation for analysis. This involves employing various techniques such as statistical methods, manual analysis, understanding protocols and data formats, linking multiple data objects, and creating timelines.

The analysis phase is inherently iterative. Initial hypotheses regarding data containing potential evidence guide the analysis, but new hypotheses may emerge during the process, prompting the collection of additional data objects. Investigations progress iteratively until results are deemed sufficient for the investigation's purpose, although achieving absolute certainty may be practically unfeasible in many cases.

- **Presentation**

This stage involves the conclusive documentation and presentation of investigation results, whether to a court of law or other relevant audiences, such as a corporation's top management or crisis management team. The presentation relies on objective findings derived from a thorough analysis of digital evidence, ensuring a sufficient level of certainty. It is crucial that the findings are succinctly summarized, and all investigative actions are thoroughly accounted for and described in a manner comprehensible to the intended audience.

2.2.3 Legal principles

In the context of a digital forensics investigation, adherence to widely accepted legal principles is paramount to ensure the integrity of the process, the protection of individuals rights, and the admissibility of evidence in court. The following legal principles should be carefully observed throughout the digital forensics investigation:

- **Data minimization**

The principle of "Data minimization" is very important to consider while

investigating a digital forensics case. According to [33], investigators should identify the minimum amount of personal data they need to fulfil their purpose. They should hold that much information, but no more. The accountability principle means that they need to be able to demonstrate that they have appropriate processes to ensure that they only collect and hold the personal data they need.

- **Proportionality**

According to Interpol [35], ensuring that the actions taken by the investigators and the methodology they used to gather evidence needs to be fair and proportionate to the interests of justice.

The prejudice (i.e. the level of intrusion or coercion) caused to the rights of any party should not outweigh the probative value of the evidence (i.e. its value as proof).

2.3 Criminal Use of Encryption & Countermeasures

2.3.1 The encryption debate

Europol published in 2015 the “**Internet Organised Crime Threat Assessment**” [11] that discusses the proposed solutions for tackling the encryption misuse problem and their limitations.

- **Outlaw encryption for general use**

There have been various proposals over the years to combat the criminal abuse of encryption. The first and most straightforward solution is to outlaw the use of encryption by citizens, some argue that ordinary citizens should not need to use encryption if they have nothing to hide while others state that the citizens right to privacy should not matter if it degrades national security and/or impedes criminal investigations.

However, these two arguments are flawed. Nowadays encryption is out of the hands of the governments. It does not require any special infrastructure and most encryption algorithms are publicly available. Making encrypted communications illegal will only hurt law abiding citizens, criminals could implement their own or use an available illegal encrypted communication service. Another counter-argument is that cyber related crime often spans multiple legal jurisdictions, sometimes even outside the EU and there is no guarantee that other countries would introduce a ban on encryption as well. Therefore, an outright ban on encryption would mostly hurt citizens and would be a very easy obstacle for criminals to overcome.

- **Key escrow**

Instead of banning encryption for everyone, the concept of **key escrow**

was introduced. Key escrows would be either government agencies or third parties entrusted with storing and securing private keys. Anyone that would want to use any form of encryption would be first required to submit their private key in the escrow. Then law enforcement agencies would be able to access any encrypted data related to a criminal investigation by submitting a warrant to the key escrow. However there are some problems with this approach that make it infeasible.

1. **Forward secrecy**

Most modern implementations of encryption employ “**forward secrecy**”. With forward secrecy the private key changes every session. Thus making it impossible to access plaintext data with a key that was submitted before the session created.

2. **Compliance**

Verifying that every key used for encryption is submitted to the escrow is impossible. The only way a LEA could make sure that all the keys of a user are validly submitted to the escrow system is when they need to access their data.

3. **Easy targets for hackers**

Key escrows are essentially databases, and therefore are prime targets for cybercriminals. There are multiple reports of database breaches each year [69]. A key escrow breach could potentially have a severe impact to all the users affected by the breach. The threat actors would be able to access all the encrypted data of their victims, that may include social networks accounts, cloud storage and even bank accounts.

4. **Jurisdiction issues**

Jurisdiction is another key challenge, one law enforcement agency in one country may require access to keys that are stored in another country. This would require international agreements and cooperation, which becomes more difficult if one of the countries involved is outside the EU.

- **Weakened encryption**

It has been suggested that only encryption that can be ‘cracked’ by law enforcement agencies should be allowed. That means that encryption keys might be limited to a certain size that it’s feasible to be brute-forced by supercomputers, or that software vendors might be compelled to introduce backdoors in their software and disclose them to governments so they can workaroud encryption.

The issue is that what is available to law enforcement agencies is also available to cybercriminals and opposing nation state actors. Backdoor vulnerabilities get discovered and cybercriminal financing grows, allowing them

access to more specialized hardware comparable to that available of most LEAs.

Clipper Chip was promoted by the US government, included a backdoor allowing access for US law enforcement and intelligence agencies. By the end of 1996 it was already defunct due to lo backlash from the US public ⁵, low adoption rate from vendors and attacks that demonstrated the backdoor could be exploited by malicious actors [24].

- **Obligation to disclose**

Another proposed solution is to make suspects criminally liable if they refuse to hand over their encryption keys to the authorities when they are investigated like refusing a breathalyzer test when driving. However this solution would not work when the keys are transient and not on the suspect's device or the encryption is seamless to the user or the suspect is unable to disclose the keys.

For example, in 2015 the FBI wanted to access the iPhone of a dead terrorist responsible for the 2015 attack in San Bernardino [18] to see if he and his collaborators planned another attack. Only the dead terrorist knew the PIN code that unlocked the device, which then sparked a long public legal battle between the Department of Justice and Apple.

Governments could also reach an agreement with the software vendors to implement security architectures that do not enable end-to-end encryption. Keys can be made accessible for LEAs through a legal/warrant request to the service provider, to hand over the encryption keys or decrypted application data of the user. The issue is that software vendors serve an international audience and enforcing no end-to-end encryption will impact users outside the EU/jurisdiction. Companies might prefer to leave the market than deteriorate the security of their products [47].

2.3.2 Encryption workarounds

Due to the limitations of the aforementioned solutions and the usefulness of encryption to safeguard people's right to privacy, it has become clear that new methods of working around encryption should be developed. Kerr and Schneier [39] have introduced a taxonomy of possible ways to work around encryption. Those categories are finding the key, guessing the key, compelling the key, exploiting a flaw in the encryption scheme, accessing plaintext and locating a plaintext copy stored elsewhere.

- **Find the key**

One method for a Law Enforcement Agency to decrypt data involves locating

⁵Article from the "Wired" against the adoption of Clipper chip, available here: <https://www.wired.com/1994/09/clipping-clipper-matt-blaze/>

an existing copy of the encryption key. This copy might take the form of a physical item, like a post-it note hidden in the suspect's room, or a digital version stored in a keystore (e.g. a password manager).

- **Guess the key**

While random encryption keys are typically long enough to render brute-forcing practically impossible, the passwords, passcodes, and passphrases used to protect these keys are often more limited in length. Easily memorable and typable, these passwords can be susceptible to guessing⁶. Additionally, users tend to reuse passwords, therefore they might be obtained from other known sources, for example previous database breaches[71]. Given that the password serves as the gateway to the encryption key, which in turn decrypts the protected data, successfully guessing the password is tantamount to obtaining the encryption key and accessing the encrypted data.

- **Compel the key**

A third strategy involves the LEAs compelling the key from an individual who possesses or is aware of it. In many instances, the crucial key is a password or passcode. Broadly speaking, compelling a key could encompass any form of coercion. In authoritarian regimes or criminal contexts, coercion might involve threats, bribery, seduction, or even torture⁷.

- **Exploit a flaw**

In such a scenario, access is obtained without needing the key, exploiting a vulnerability in the system designed to prevent unauthorized entry. Various forms of weaknesses can be exploited. These may include a flaw in the mathematical foundation of the encryption algorithm, a vulnerability in the random-number generator supplying inputs to the encryption algorithm, or weaknesses stemming from the implementation of the algorithm in software on a computer. One such example was when the FBI used an exploit chain to access the iPhone of a deceased terrorist at San Bernardino in 2015 [18].

- **Access plaintext while in use**

Intended end users cannot read ciphertext, therefore encrypted data must be decrypted to be read by them. This creates a security gap that LEAs can leverage to gain access to the decrypted data. The technical complexity of these methods varies, from grabbing the device while it is unlocked and used by the suspect [5, 7] to advance techniques such as the 0-day exploit that was used by the FBI to reveal the IP address and apprehend “Brian Kil” [23].

⁶Cupp See: github.com/Mebus/cupp

⁷See relevant XKCD comic about compelling encryption keys: xkcd.com/538/

- **Locate a plaintext copy**

Another strategy is to obtain a separate and unencrypted copy of the information. The target might possess multiple copies of the desired records, and LEAs could potentially access an unencrypted version. This method involves searching for an alternative copy of the sought-after information. In this context, this approach may be viewed more as an alternative strategy than a workaround. Rather than bypassing encryption, it entirely avoids the encryption process. Given the popularity of cloud services, the FBI, for instance, has established a legal framework to access backup data for various applications stored in the cloud ⁸.

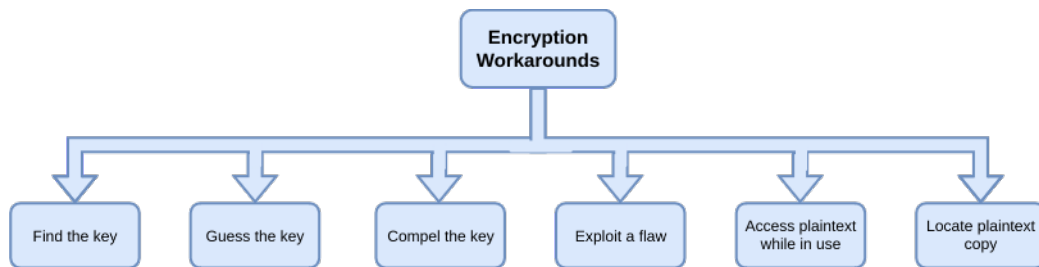


Figure 2.5: Kerr & Schneier’s encryption workarounds taxonomy.

One such successful case of encryption workaround is the “**EncroChat**” case. EncroChat, was a service that provided secure phones to criminal organizations. Authorities in Europe and especially in France often encountered these devices in their investigations but were unable to decrypt them due to the sophisticated security measures of the devices. Eventually, the French authorities with the help of Netherlands and Europol managed to break the encryption security mechanisms of EncroChat by infiltrating the service and installing a malware able to intercept and decrypt the criminals’ communications. This operation, lead to the dismantle of the EncroChat service and numerous arrests across Europe[17].

Europol in [11] states that LEAs, researchers and industry should work together to develop a framework that enable working around encryption. This methods should leverage weaknesses in the implementation and usage of cryptographic primitives like symmetric and asymmetric algorithms and also attacking hashes

⁸A page from an FBI training document was made public in 2021 depicting various applications and what data can the FBI get from cloud backups. [propertyofthepublic.org/document-detail/?doc-id=21114562](https://www.propertyofthepublic.org/document-detail/?doc-id=21114562). A blog by MalwareBytes discusses it in more detail. www.malwarebytes.com/blog/news/2021/12/heres-what-data-the-fbi-can-get-from-whatsapp-telegram-and-more

whenever possible. Using the aforementioned methods when a legal basis has been established and respecting citizens rights, we focused our research on finding encryption workarounds that would enhance the capacity of LEAs to successfully access encrypted data. Next chapter introduces the encryption workarounds we found. For a categorization of the encryption workarounds, refer to Appendix A.1.

Chapter 3

Methodology

In this chapter we present a compilation of encryption workarounds we found online targeting various software applications. We split the workarounds in two categories based on the type of the file that is being analyzed, either **file system** or **memory**, then for each software we explain how it is affected by the workaround in detail.

First, we give a brief overview of the workaround method by introducing the targeted software. Then in the **Architecture** section, we discuss in detail the architecture of the targeted software, how it handles encryption and/or credentials storage and possible methods that can mitigate the threat of the workaround. In the **Scenario** section, we discuss how it would benefit LEAs to leverage the workaround by linking the workaround method with the high level taxonomy introduced by Kerr and Schneier[39]. In the **Setup** section, we demonstrate the workaround along with the prerequisites needed to run the workaround. Finally, in the **Validation** section we validate that the method is forensically sound using digital forensic best practices so it can be used effectively by LEAs by being admissible in a court of law.

Note, that the workarounds presented in this section were not implemented by us. Our contribution is (i) the mapping between the technical encryption workarounds to the higher level overview of Kerr et al.[39], (ii) the testing of the modules, and finally (iii) the validation of their forensic soundness in the context of digital forensics.

While the process of device acquisition is crucial in the field of digital forensics and demands careful handling, it falls beyond the scope of this thesis. We made every effort to adhere to best practices and procedures, but it's important to note that we lacked access to resources like hardware write blockers and imaging devices.

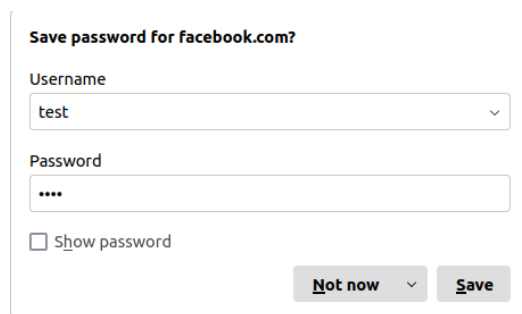
3.1 File system analysis

File system forensics plays a pivotal role in digital investigations by providing access to a wealth of information stored on a computer or storage device. Understanding the file system is essential for data recovery, evidence preservation, artifact identification, timeline reconstruction, and malware analysis, making it a cornerstone of digital forensics practice. Below, we list three cases we explored for bypassing encryption through file system analysis. These methods are for the following software:

1. Mozilla software such as the Firefox browser , Thunderbird email client and TOR browser
2. Google Chrome and similar browsers Microsoft Edge and Opera
3. Meo encryption software

3.1.1 Mozilla software

Firefox developed by Mozilla is one of the most known browsers, according to their statistics page¹ it is used by 362 million people across the globe to access the world wide web. **Firefox** and other mozilla based software such as **Thunderbird** and **TOR browser** offer the functionality to save and remember login credentials for websites and email accounts. When a user visits a website and fills a credentials form they will be asked whether they prefer the browser to remember the credentials next time they visit the website as seen in Figure 3.1. If the user agrees, Firefox will save the credentials (username & password) locally in the user's machine along with the corresponding url of the website.



The image shows a dialog box titled "Save password for facebook.com?". It contains a "Username" field with the value "test" and a "Password" field with masked characters "****". Below the password field is a checkbox labeled "Show password" which is currently unchecked. At the bottom right, there are two buttons: "Not now" with a dropdown arrow and "Save".

Figure 3.1: Mozilla Firefox prompt to save web credentials when logging in a website.

¹See <https://www.enterpriseappstoday.com/stats/firefox-statistics.html>

3.1.1.1 Architecture

Mozilla software encrypts the remembered credentials before storing them locally, but the encryption key is saved as a plaintext in the user's machine. Over the years, there were various methods that Mozilla software handled credential storage differently, but the encryption key was always stored as plaintext. Therefore, the same credential recovery method can be leveraged in all versions (newest version: 121.0) as of the current writing of this thesis. In the next paragraph we first describe the current process of credential storage and in second paragraph we compare it to the older versions.

How Firefox stores passwords:

Mozilla encrypts both usernames and passwords using **AES-CBC** and stores them in the `logins.json` file (refer to Listing 1), along with other information related to each credential pair. The encryption key used is derived using the "Password-Based Key Derivation Function 2 (PBKDF2)" [53], which requires a password to generate an encryption key. This password is a combination of the SHA256 output of the "Global Salt" value defined by Mozilla, along with a user-supplied password, which is optional and empty by default. The "Global Salt" is stored in the `key4.db` file, along with other data needed for encryption, such as the Initial Vector (IV), encryption key length, and iteration count, which are encoded using **ASN.1** [2] before being placed inside the file. This whole process is illustrated in Figure 3.2. Finally, for a detailed explanation for the rest of the fields and their forensic value in Listing 1 see Nelson et al. [50].

Listing 1 Example output of `logins.json`

```
{
  "id": 2,
  "hostname": "https://www.facebook.com",
  "httpRealm": null,
  "formSubmitURL": "https://www.facebook.com",
  "usernameField": "email",
  "passwordField": "pass",
  "encryptedUsername": "MDoEEPgAAAAA...BBBTY9R5lmpOMv4kv1VHnBm",
  "encryptedPassword": "MDoEEPgAAAAA...BBDZF1rgDU0zEMXjC0a2rxDY",
  "guid": "{0256321d-c4b7-4969-b66e-d030d6ba2027}",
  "encType": 1,
  "timeCreated": 1661366966952,
  "timeLastUsed": 1661366966952,
  "timePasswordChanged": 1661366966952,
  "timesUsed": 3
}
```

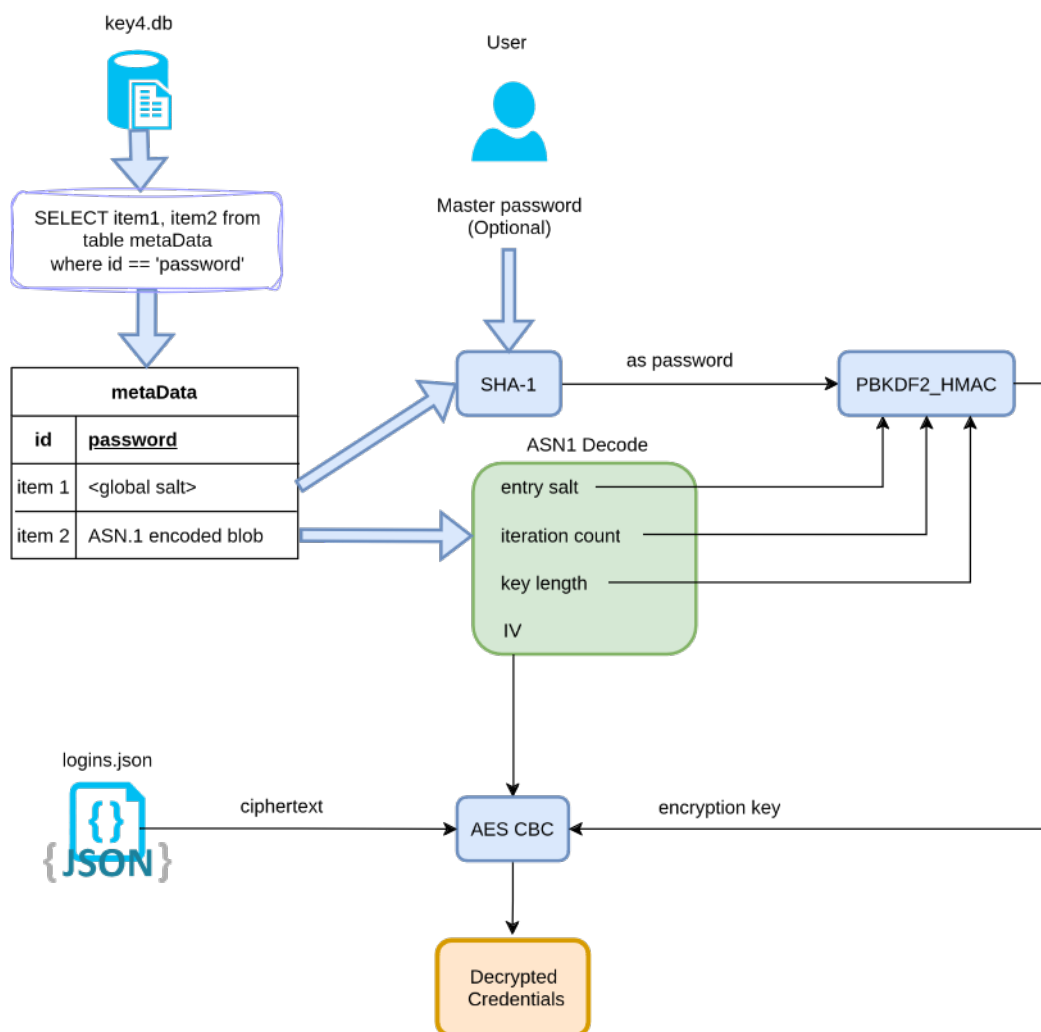


Figure 3.2: Decryption process of saved user web credentials in Mozilla software such as Firefox.

Older Versions: Previous versions of Mozilla software followed the same principle for storing credentials. The only thing that changed was the type of encoding, encryption and the file names². Notice that instead of `key4.db` and `logins.json` files it has `keys3.db` and `signons.sqlite` instead. But the methodology remains the same, all versions up to the newest in time of testing, which is version 121.1 (released January 9, 2024).

Mitigations: Decrypting user passwords will work in most cases since the

²For previous Firefox versions see github.com/lclevy/firepwd/blob/master/mozilla_pbe.svg.

method of storing the encryption key for the credentials in plaintext form existed in all the versions that supported credentials storage. If a user sets a browser **master password** to access the credentials then this method will not work. The master password which is not stored in the device, is used to derive the encryption key that encrypts the data, without it the decryption will fail. However, this is not the default option in all the above-mentioned software and must be explicitly enabled by the user.

If a master password is set, the investigators would have to know or obtain the password in order to gain access to the web logins. One such way is by performing bruteforce attacks if the password is relatively short. Another way is to guess the password by trying popular passwords or passwords that the user might use in other services, due to the fact that password reuse is popular among users [9].

3.1.1.2 Scenario

This method can be categorized as “**Finding the key**” in Kerr & Schneier’s Taxonomy which generally do not lead to human rights problems [39]. This workaround enables LEAs to find credentials for a suspect’s online accounts that are saved in the browser. They may provide useful information for investigators such as online communication via messaging apps, photos and other files from cloud backups. Additionally, the recovered passwords might grant access to other accounts since users commonly reuse their passwords across various online services [9, 21].

3.1.1.3 Setup

The experiment was performed both in Windows and Linux Operating Systems. Both OSes were installed in a Virtual Machine. For the Linux OS we used *Kali Linux x64 version 2021.1*. For Windows we used *Windows 10 x64 version 20H2*. In both cases the next steps are similar the only thing that changes is the path of the default profiles, therefore we only show the windows version.

The default location paths for the profiles on Windows and Linux OSes are the following:

- Windows: `C:\Users\\AppData\Roaming\Mozilla\Firefox\Profiles`
- Linux: `~/.mozilla/firefox`

Tools: We used `firefox_decrypt` that retrieves the passwords of a Mozilla profile if no master password is set. The tool requires python 3 to be installed in the target system, with older versions using python2. However, we stucked to the newest available version at the time and used `python 3.10.6` for Windows.

Installation:

- Python version 3.10.6: downloaded from <https://www.python.org/downloads>

- Git (GitBash): downloaded from <https://gitforwindows.org/>, if you are on Linux git is included by default on most distros.
- `firefox_decrypt`: to install the script open a terminal or GitBash if you are on Windows and type:

```
git clone git@github.com:unode/firefox_decrypt.git
```

Target Software We installed Mozilla software that uses the aforementioned process of storing and accessing credentials. Such tools are:

- **Firefox:** Version 121.1 (64-bit), released January, 2024.
- **Thunderbird:** Version 115.6.0 (64-bit), released December, 2023
- **TOR Browser:** Version 13.0.9 (based on Mozilla Firefox 115.7.0esr)(64-bit), released Jan, 2024

1. Firefox

There is no additional configuration needed for Mozilla Firefox to store and access passwords. We manually created new dummy logins by navigating to the Passwords section of Firefox (Figures 3.3 and 3.4).

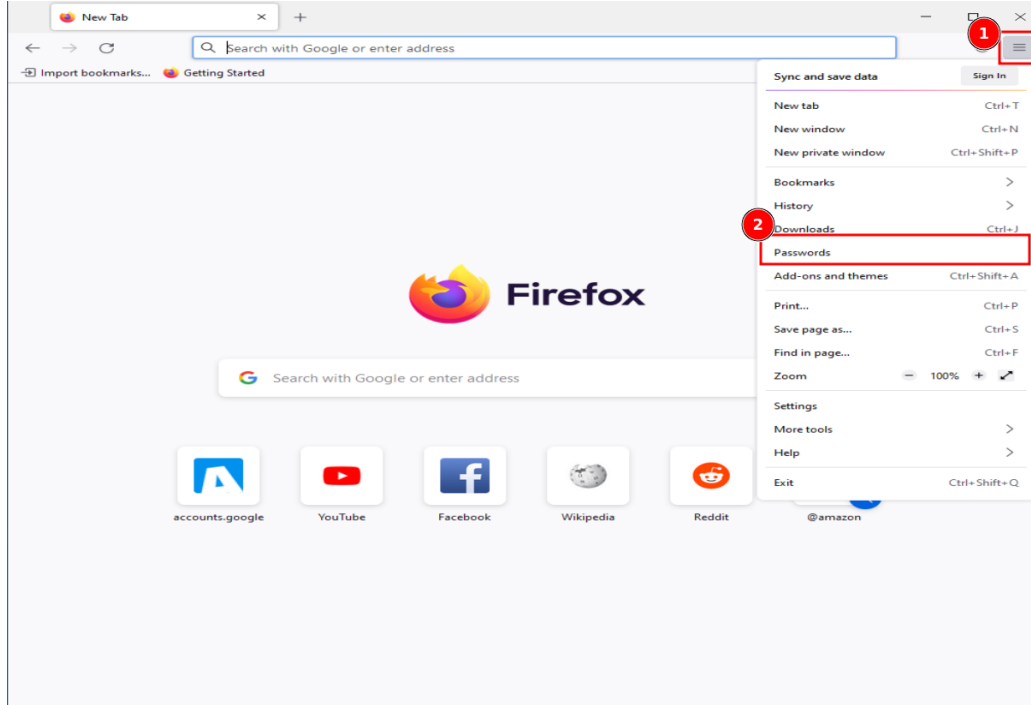


Figure 3.3: How to navigate to the Firefox saved logins and passwords page.

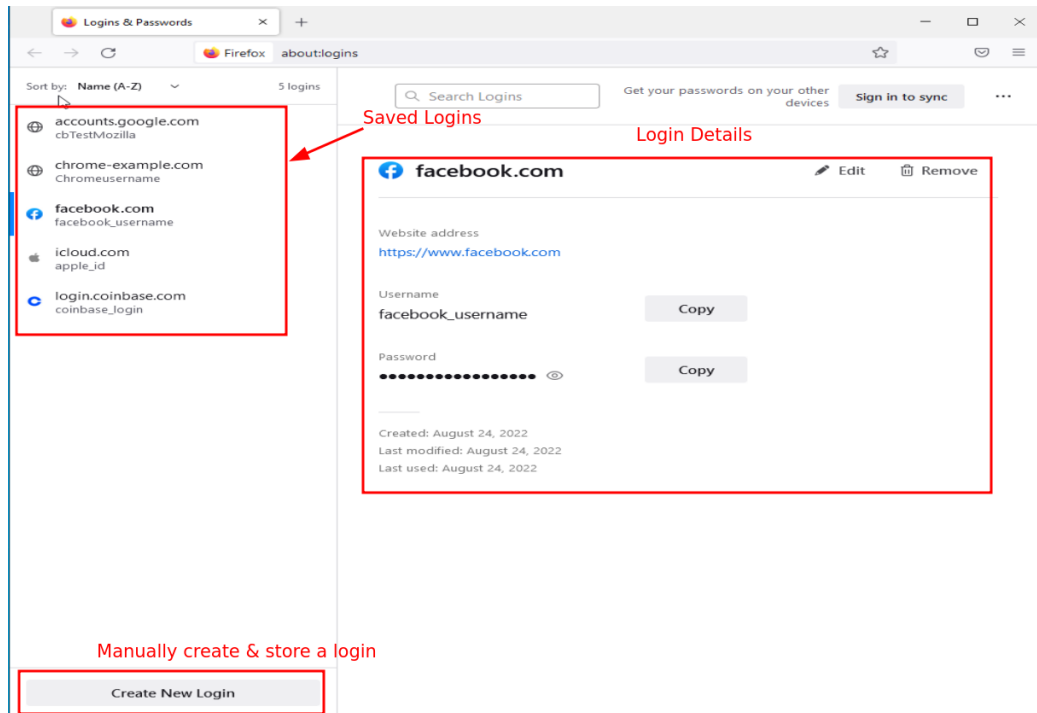


Figure 3.4: Firefox Passwords page.

We run the Firefox Decrypt³ to obtain the passwords. We get prompted to select a user profile. Depending on the number of users that use the same computer we may have more than 2 profiles to select from. A user can create as many different profiles in Firefox that have different configurations, preferences and history (including passwords and cookies). However, in the case of one user with one profile we will be presented with only 2 profile options to select. The correct profile is always the option 2 because it is the **default-release** profile. The output (Figure 3.5) contains a list of usernames and plaintext passwords along with the websites they are being used for.

```
py firefox_decrypt.py
```

³Firefox Decrypt: available at github.com/unode/firefox_decrypt, credits unode

```
dionkal ~/main ... 3 ~ > repos > firefox_decrypt > python3 firefox_decrypt.py 6oz90sz2.default-release/
2024-02-14 14:22:19,239 - WARNING - profile.ini not found in 6oz90sz2.default-release/
2024-02-14 14:22:19,239 - WARNING - Continuing and assuming '6oz90sz2.default-release/' is a profile location

Website: https://www.facebook.com
Username: 'facebook_username'
Password: 'facebook_password'

Website: https://www.icloud.com
Username: 'apple_id'
Password: 'apple_password'

Website: https://login.coinbase.com
Username: 'coinbase_login'
Password: 'coinbase_password'

Website: https://accounts.google.com
Username: 'cbTestMozilla'
Password: 'gmail password'
```

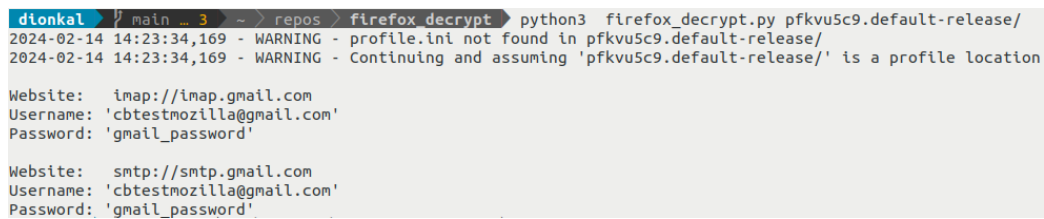
Figure 3.5: Firefox decrypted passwords.

2. Thunderbird

We used the same exact process as we did with Firefox. The only difference is the location of the default profile. Firefox Decrypt by default uses the Mozilla Firefox profiles save location, which is different depending on the OS. The tool accepts as a second optional argument a path to a different location to search for Mozilla profiles.

```
py firefox_decrypt.py <path_to_thunderbird_profile>
```

Invoking the tool with the default location where thunderbird saves the profiles will get return a list of credentials used by thunderbird, those are mainly email accounts as seen at Figure 3.6.



```
dionkal ~$ python3 firefox_decrypt.py pfkvu5c9.default-release/
2024-02-14 14:23:34,169 - WARNING - profile.ini not found in pfkvu5c9.default-release/
2024-02-14 14:23:34,169 - WARNING - Continuing and assuming 'pfkvu5c9.default-release/' is a profile location

Website: imap://imap.gmail.com
Username: 'cbtestmozilla@gmail.com'
Password: 'gmail_password'

Website: smtp://smtp.gmail.com
Username: 'cbtestmozilla@gmail.com'
Password: 'gmail_password'
```

Figure 3.6: Thunderbird decrypted passwords.

3. TOR Browser

Decrypting passwords stored in the TOR Browser follows a similar process to thunderbird. The location of the TOR profile must be supplied to Firefox Decrypt. However, there is one notable difference from Thunderbird, the TOR Browser is designed with strict privacy and security controls in mind. The developers have disabled TOR from storing and remembering passwords by default. A user must manually turn this feature on from the settings.

To enable storing passwords, a user must navigate to the TOR Browser settings page as shown at Figure 3.7, click the Privacy & Security tab on the left. Then select the “*Use custom settings for history*” history dropdown, enable (i) Remember browsing and download history (ii) Remember search and form history options, depicted in Figure 3.8. After these changes, TOR will ask the user every time they login to a website if they want the TOR Browser to remember and auto-fill passwords.

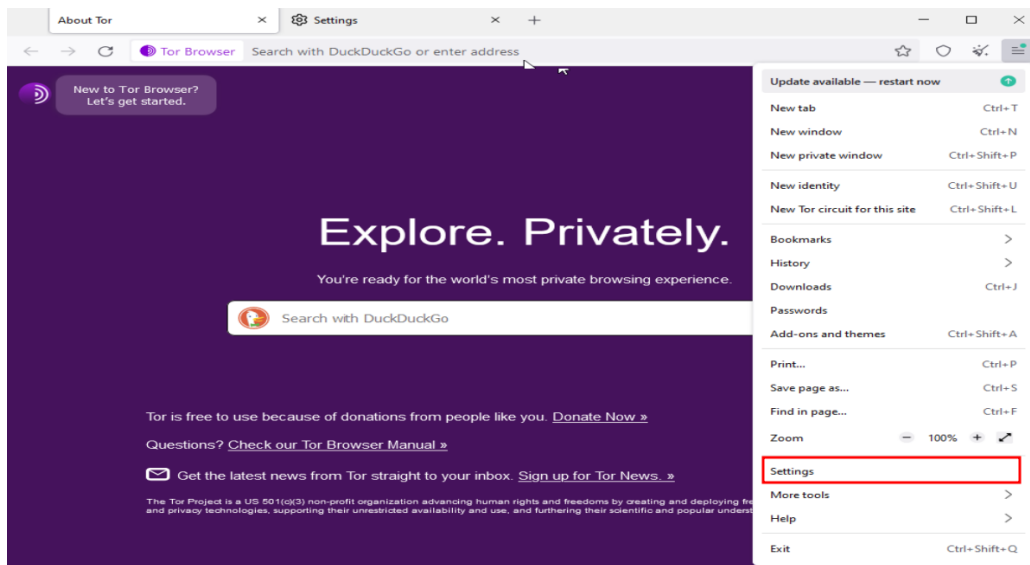


Figure 3.7: Open TOR Browser settings.

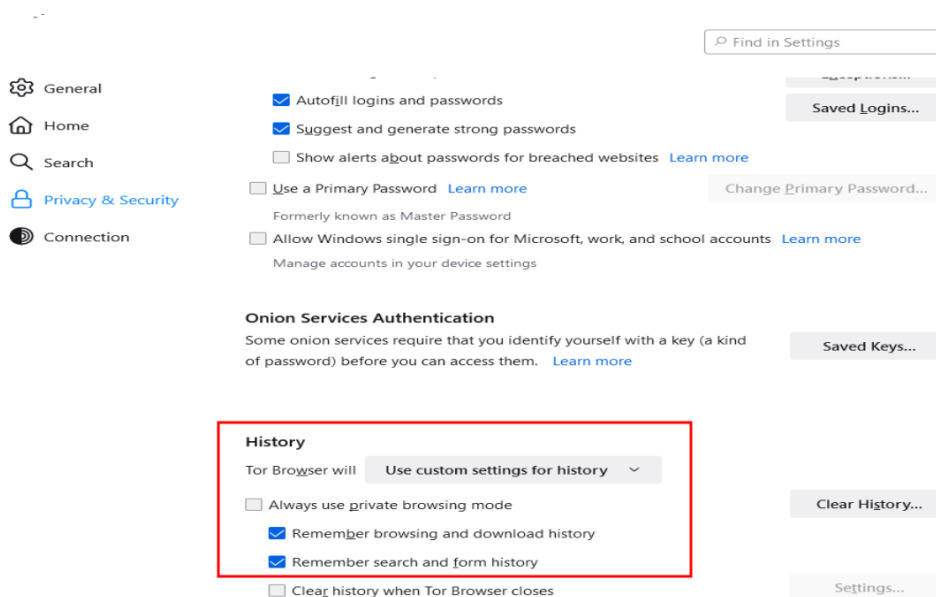


Figure 3.8: Settings that need to be enabled to allow TOR to store passwords.

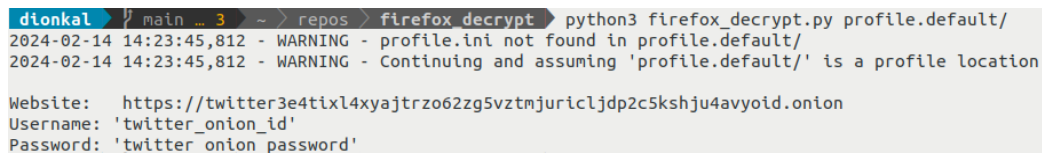
For the experiment we manually added a dummy entry in the password manager as we did with Firefox. The credentials we added for the twitter onion website were:

- username: twitter_onion_id

- password: twitter_onion_password

We then run the Firefox Decrypt tool by providing the path to the TOR profile and got the username and plaintext password along with the url of the twitter onion website.

```
py firefox_decrypt.py <path_to_TOR_profile>
```



```
dionkal ~$ python3 firefox_decrypt.py profile.default/
2024-02-14 14:23:45,812 - WARNING - profile.ini not found in profile.default/
2024-02-14 14:23:45,812 - WARNING - Continuing and assuming 'profile.default/' is a profile location

Website: https://twitter3e4tixl4xyajtrzo62zg5vzmtjuricljdp2c5kshju4avvoid.onion
Username: 'twitter_onion_id'
Password: 'twitter onion password'
```

Figure 3.9: TOR decrypted passwords.

3.1.1.4 Validation

Firefox decrypt by default searches for mozilla profiles in the current system. However, in the context of an investigation, the examiner will run the workaround from their workstation and will take as input the location of the suspect’s user profile. The workaround will not alter in any way or form the source. Nevertheless, it is advised to use a forensically acquired physical or logical extraction of the suspect’s file system.

To validate the preservation of the source, we compared its hash signature from before and after running the workaround. Since there are two files involved in the key storage and decryption process (`logins.json`, `key4.db`), we calculated the hashes for both files using MD5⁴ and SHA256⁵. Table 3.1 shows that that the `logins.json` file did not change and Table 3.2 shows that the `key4.db` file did not change as well.

	MD5
before	e87ffdb014a559182377892015746759
after	e87ffdb014a559182377892015746759
	SHA256
before	2e60129ed5b1bd470a0d926526a3ae3991dd7b49d8322bd8994dd44b36d53ee6
after	2e60129ed5b1bd470a0d926526a3ae3991dd7b49d8322bd8994dd44b36d53ee6

Table 3.1: MD5 & SHA256 hashes of “`logins.json`” before and after running the decryption workaround

⁴See <https://www.man7.org/linux/man-pages/man1/md5sum.1.html>

⁵See <https://www.man7.org/linux/man-pages/man1/sha256sum.1.html>

MD5	
before	bf1c07bde40ab1223079ff7b28d6ae04
after	bf1c07bde40ab1223079ff7b28d6ae04
SHA256	
before	02a31a29edf7602e712448715ca0424997b5ae6184aaacb1a62d7e4c39b736b9
after	02a31a29edf7602e712448715ca0424997b5ae6184aaacb1a62d7e4c39b736b9

Table 3.2: MD5 & SHA256 hashes of “key4.db” before and after running the decryption workaround

3.1.2 Chrome Decrypt

Google Chrome is the most used browser in personal computers ⁶. Like Firefox, Chrome offers the functionality to save and remember user credentials including passwords to online accounts. Figure 3.10 shows the prompt that Chrome displays when a user logs in to a website that hasn't logged in before with those credentials. If the user clicks the "Save" button then chrome will remember the credentials used and will prompt to auto-fill them next time the user visits the same website. Last but not least, **Microsoft Edge** and **Opera** browser follow the process of storing and retrieving web credentials, therefore the same encryption workaround described below can be leveraged to obtain plaintext credentials.

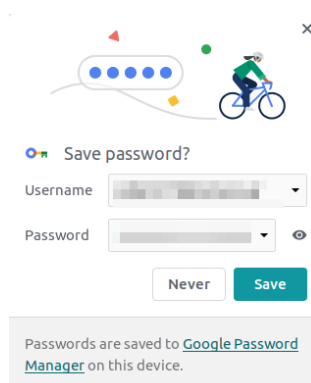


Figure 3.10: Google Chrome prompt to save credentials

⁶Chrome had the 66,18% of the Desktop browser marketshare from Nov 2021 to Nov 2022, <https://www.enterpriseappstoday.com/stats/desktop-browser-statistics.html>

3.1.2.1 Architecture

logins table	
field	type
origin_url	VARCHAR
action_url	VARCHAR
username_element	VARCHAR
username_value	VARCHAR
password_element	VARCHAR
password_value	BLOB
submit_element	VARCHAR
signon_realm	VARCHAR
date_created	INTEGER
blacklisted_by_user	INTEGER
scheme	INTEGER
password_type	INTEGER
times_used	INTEGER
form_data	BLOB
display_name	VARCHAR
icon_url	VARCHAR
federation_url	VARCHAR
skip_zero_click	INTEGER
generation_upload_status	INTEGER
possible_username_pairs	BLOB
id	INTEGER
date_last_used	INTEGER
moving_blocked_for	BLOB
date_password_modified	INTEGER

Table 3.3: List of fields and their respective types of the logins table that Chrome uses to store passwords

Chrome manages a local sqlite file called **Login Data** in the user's computer. The file contains information that the chrome password manager collects. More specifically, the Table 3.3 contains the user's credentials that Chrome manages and autofills on behalf of the user.

- **origin_url**: Is the url that the credentials belong so that Chrome can detect the correct credentials and use them in the correct site. It is of type **VARCHAR** because it stores strings.
- **username_value**: Is the username for the specific account that belongs to this url. It is of type **VARCHAR** because it stores strings.
- **password_value**: Contains the encrypted password for this url. Notice, that it is of type **BLOB** which is used for storing binary data⁷. Additionally, the encryption key that decrypts the Login Data file is further secured by relying on the underlying operating system as seen in Figure 3.11.

⁷Read more about BLOB type, <https://sqldocs.org/sqlite/sqlite-blob/>

Other fields can also have forensic value, refer to Chapter 3.1.2.2 for more details.

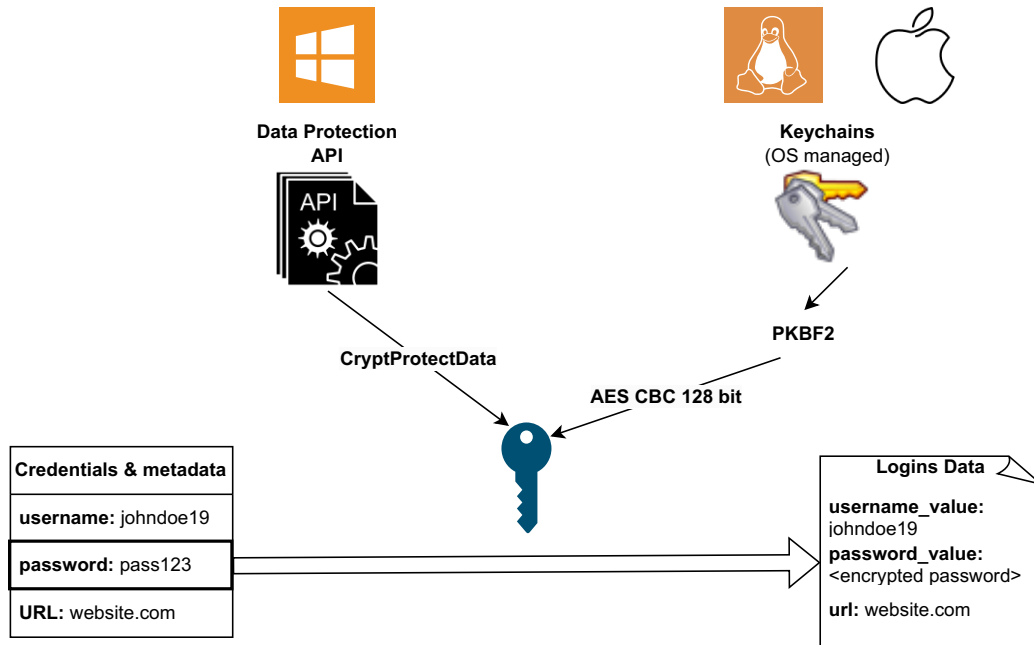


Figure 3.11: Password encryption process in Chrome

Windows: Chrome encrypts the passwords stored in the **Login Data** by calling the `CryptProtectData` function of the **Windows Data Protection API (DP API)**. Similarly, in order to decrypt the password it calls the `CryptUnprotectData` function from the Windows API [44]. The documentation mentions that decrypting data protected with `CryptProtectData` is only possible if performed by the same user on the same computer, with two notable exceptions—neither of which is used by Chrome:

1. **Roaming user:** A user with a roaming profile can decrypt their data from another computer in the same network. Usually, this type of profile is encountered in corporate environments.
2. **Protect in local machine** If the flag `CRYPTPROTECT_LOCAL_MACHINE` has been enabled when the `CryptProtectData` was called then every authenticated user in the same computer can decrypt the data.

DPAPICK: Additional independent research [10] has shown that it is possible to decrypt DPAPI protected data offline **provided** that you know the **Windows user password**. This method is called DPAPICK and works for local Windows

accounts (no active directory nor Microsoft Live) and without TPM protection enabled. The way DPAPI works with Chrome is that it encrypts the key that decrypts the passwords in the `Login Data` file and it creates a DPAPI blob. A DPAPI blob consists of two parts, (i) the encryption & decryption keys (DPAPI master keys) and (ii) the encrypted data itself.

- **Master keys:** Master keys can be of two types:
 1. System: Every system has System Master Keys that are used to encrypt data that must be accessible for every user. These can be found in the folder:


```
%windir%\System32\Microsoft\Protect\S-1-5-18\User
```
 2. User: Unique to each user and stored in


```
%APPDATA%\Microsoft\Protect
```

 within a folder that is the user Security Identifier (SID).
 This specific category of master keys is utilized by Chrome to ensure that the passwords of a single user are inaccessible by all the other users.

These Master Key files are also binary, hex-formatted data structures holding several fields, however the DPAPI Master Key itself is encrypted too and needs a user password (hash) and the user SID (which is the folder name). This User Master Key can be decrypted using the user SID and the SHA1 hash of the clear text user password (UTF-16LE encoded).

MacOS/Linux: In MacOS and Linux the passwords are encrypted using AES-128-CBC. AES is a symmetric block cipher algorithm that uses a key and an Initialization Vector (IV) to both encrypt and decrypt data. In this instance, the iv is 16 bytes of the character space (0x20) while the key is generated by the Password-Based Key Derivation Function 2 (PBKDF2) [53].

In order to derive the encryption key from the PBKDF2 Chrome generates a random password and stores it in a keychain (such as Gnome keyring, KDE keyring, MacOS Keychain Access, etc.) managed by the operating system (e.g. Linux Pluggable Authentication Module)⁸. We chose to prioritize our research on unlocking Windows secrets as it promised the most significant impact, although we did not concentrate on Linux and MacOS. Nevertheless, exploring this aspect remains part of our future agenda.

3.1.2.2 Scenario

This method of workaround falls under the same category in Kerr's & Schneier's taxonomy [39] as the workaround for the Mozilla password manager which is "Finding the key". Although the data we are after are encrypted, we are able to obtain

⁸See "An introduction to Pluggable Authentication Modules (PAM) in Linux" by Redhat, <https://www.redhat.com/sysadmin/pluggable-authentication-modules-pam>

the decryption key from the operating system and decrypt them provided that we know the Windows user password. Decrypting the database may yield the user's credentials to various online accounts such as end to end encrypted messaging applications, cloud backups, online forums etc., that may be valuable in the scope of a criminal or civil investigation.

Further analysis of the Login Data database may be fruitful. In [22] the author describes how Chrome interacts and populates the Login Data database and what information can be obtained from various fields. For example, whenever a user clicks the 'Never' option in the Chrome prompt (Figure 3.10), it still creates an entry in the logins table with the `website_url` and the `blacklisted_by_user` flag set to 'true' (see Table 3.3). This can reveal if a user attempted successfully or not to login to a particular site.

The field `times_used` counts the times Chrome auto-filled the password in the login form of the website on behalf of the user and therefore can reveal the times the user has logged in that particular site.

Lastly, if the user doesn't perform any action in the Chrome prompt - Save or Never -, Chrome creates an entry in the stats table with the website domain, username & update time. Therefore, an investigator can learn the websites that the user tried to log in with a particular username along with the last time he tried.

3.1.2.3 Setup

To evaluate this encryption workaround using the DPAPICK technique, we employed the following methodology. First we setup a virtual machine with all the required software, then we performed an acquisition to get the virtual machine to ewf format and then run the script to perform the encryption workaround. It is important to note that apart from Google Chrome we also performed the same experiment to Microsoft Edge and Opera browser as well, due to the fact that they utilize the same method to protect their user passwords.

Virtual Machine: As described above, we setup a virtual machine running Windows 10, and we installed the following:

- Windows 10 Pro version 22H2 build 19045.2846 (Released April 2023)
 - username: `ChromeUser`
The account is local (not connected to AD not even to a Microsoft online user account)
 - password: 123
- Google Chrome browser version 117.0.5938.89 64-bit (Released September 2023), saved credentials (not signed in to google account):

- url: `www.facebook.com`
 - username: `chromeuser@testemail.com`
 - password: `chrome_facebook_password123`
- Opera browser version 103.0.4928.47 64-bit (Released October 2023), saved credentials:
 - url: `www.twitter.com`
 - username: `operauser@testemail.com`
 - password: `opera_twitter_password123`
- Microsoft Edge browser version 118.0.2088.76 64-bit (Released October 2023), saved credentials:
 - url: `www.binance.com`
 - username: `edgeuser@testemail.com`
 - password: `edge_binance_password123`

Acquisition & Analysis: To acquire the image of the virtual machine and mount it in our forensic workstation we used did the following:

- Convert the virtual disk image to raw:

```
$ VBoxManage clonemedium <virtual.vdi> output.img  
--format raw
```

- Convert the raw image to Expert Witness Format (ewf)⁹:

```
$ ewfacquire -t <target> input
```

- Mount¹⁰ the ewf image, :

```
$ sudo imount ewf_image.e01
```

- Extract the following files to unlock DPAPI protected blobs:

⁹Github repo of libewf which has the ewfacquire tool, <https://github.com/libyal/libewf>

¹⁰Python package for imount, <https://pypi.org/project/imagemounter/>

- Local State:
C:\Users\ChromeUser\AppData\Local\Google\Chrome\User Data\Local State
- Login Data:
C:\Users\ChromeUser\AppData\Local\Google\Chrome\User Data\Default>Login Data
- Masterkey file:
C:\ChromeUser\AppData\Roaming\Microsoft\Protect\S-1-5-21-3472447873-910604344-166312159-1001\ad489ebb-0f94-4a4c-8005-48fc871e4cf6
The MK filename will be displayed once the script is run with only the local state and login data as arguments.

- Run the decryption script¹¹:

```
$ python3 browserdec.py -t <path to local state> --loginfile <path to login data> -m <path to mk file> -p <windows user password> -s <Windows user security id> -v
```

3.1.2.4 Validation

Finally, it was important to confirm the forensic integrity of the methodology employed in the “*Setup*” section, ensuring that it did not modify the content of the extracted files. To accomplish this, we conducted a comparison between the pre-workaround **md5** and **sha256** hashes of the **Login Data**, **Local State**, and **MasterKey** files for each browser and those obtained after employing the DPAPICK method. As depicted in Tables 3.4, 3.5, and 3.6 the MD5 and SHA256 hashes of the **Login Data**, **Local State**, and **MasterKey** files, utilized by the DPAPICK technique to decrypt stored web logins in Google Chrome, demonstrate no alteration in the content of the respective files. For conciseness, the tables specifically present the validation of the DPAPICK approach for Google Chrome, while we have also verified its validity for Microsoft Edge and Opera.

¹¹dpapilab-ng browserdec.py script that extracts chrome passwords offline <https://github.com/tijldeneut/dpapilab-ng/blob/main/browserdec.py>

Login Data	
MD5	
before	8c7a7434bfb9f648db7c37ac2d4fbcd4
after	8c7a7434bfb9f648db7c37ac2d4fbcd4
SHA256	
before	6a2dd95e2a8e7674582d4179ca6e89d2e44f8837e1a2f90f377962b6ec5d3719
after	6a2dd95e2a8e7674582d4179ca6e89d2e44f8837e1a2f90f377962b6ec5d3719

Table 3.4: MD5 & SHA256 hashes of the files associated with the DPAPICK method for Google Chrome.

Local State	
MD5	
before	80822b05f234dac760c34363a754fdea
after	80822b05f234dac760c34363a754fdea
SHA256	
before	4f9178daa352340c6d48aa340a945fd0a6aeed9861f00578c00d3ff38f29ddd7
after	4f9178daa352340c6d48aa340a945fd0a6aeed9861f00578c00d3ff38f29ddd7

Table 3.5: MD5 & SHA256 hashes of the “Local State” file.

masterkey file	
MD5	
before	bc61c32cb3a99f53f872eb9cf7305f7d
after	bc61c32cb3a99f53f872eb9cf7305f7d
SHA256	
before	1f30e91aaea241cd5cad40aadb427a65d2ca0a3eddb0f822acc7f90acfbec12e
after	1f30e91aaea241cd5cad40aadb427a65d2ca0a3eddb0f822acc7f90acfbec12e

Table 3.6: MD5 & SHA256 hashes of the masterkey file.

3.1.3 Meo encryption software

Meo¹² is an encryption software published by NCH Software. It is available in both Windows (7/8/8.1/10/11) and MacOS (10.5 - 10.14) and it offers the functionality of encrypting files and emails using either Blowfish or DES encryption algorithm.

3.1.3.1 Architecture

To encrypt a file or an email, Meo requires a password given by the user. The password then is used to generate the encryption key for the equivalent encryption algorithm, Blowfish¹³ or DES (as seen in Figure 3.12). To decrypt the encrypted file, Meo requests the password from the user once again, if the password is correct it decrypts the file successfully. On the other hand, if the password is incorrect it fails decrypting the file and displays an incorrect password message.

In order for Meo to determine if the password that was given during the decryption process by the user is the correct one, it needs to know the correct password in the first place. Therefore, meo hashes the password during the encryption process and it stores it at the start of the encrypted file. In the next section we discuss in detail the process of storing the password by Meo and how we can leverage it to retrieve the password and decrypt the file.

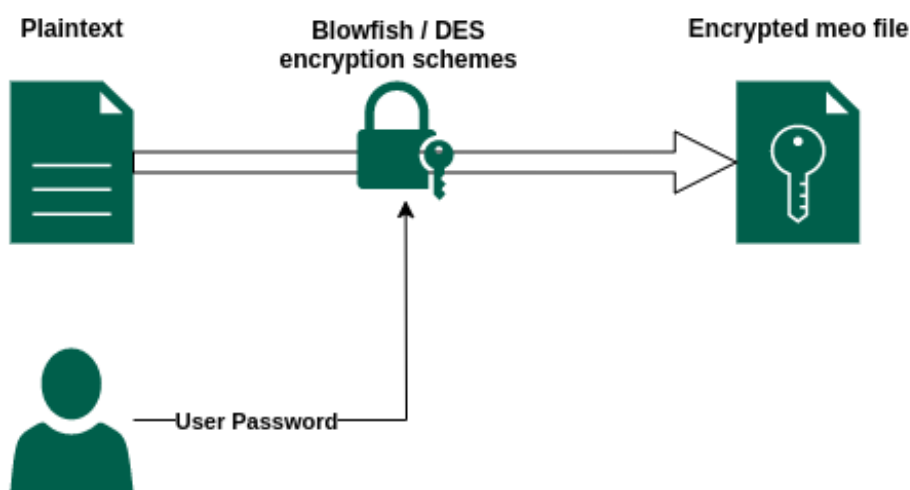


Figure 3.12: High-level overview of Meo encryption process.

Analysis: As described above, Meo stores the password used to encrypt the file in the start of the encrypted file. The author of the blog [66] has performed the initial analysis on Meo which this section focuses on but their analysis is limited

¹² Available at <https://www.nchsoftware.com/encrypt/index.html>

¹³ According to MEO FAQ, Blowfish will be 256 bit only if the provided password is large enough, we do not have any information about the size of the key for DES, <https://www.nchsoftware.com/encrypt/kb/1335.html>

only on the encryption based on the **Blowfish** algorithm. Figure 3.13 describes the process, first, the password is hashed using the MD5 hash function algorithm, next every byte of the hash value xor-ed with the value 254 (0xfe in hexadecimal format) and then it is inserted at the start of the encrypted file.

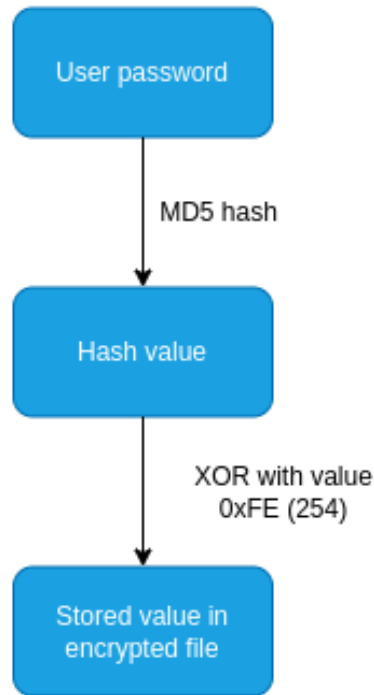


Figure 3.13: Process of user password storage.

The resulting value is stored in the first bytes of the encrypted file along with other relevant information as seen in figure 3.14. The first 9 bytes (red) contain the header that identify the file as encrypted by Meo which are always the same “ HCN 0.1”. Next, after skipping 11 unused bytes that are irrelevant, the next 4 (green) contain the length of the hashed password (in little endian format). In our testing we did not see any variation in key length, this field remains unused. The next 32 bytes (pink) are the hashed and xor-ed value of the user password that we described above.

```

20 48 43 4E 20 30 2E 31 01 00 00 00 00 00 00 00 HCN 0.1.....
00 00 00 00 20 00 00 00 CA C6 CC 9D C6 CF CF 9A .....
9F CB 9A CB 9C CA 9C 9D C8 9A CA C7 C9 98 98 9F .....
C7 C6 CA C7 CF 9B CD C6 00 00 00 00 2E 00 00 00 .....
AA A4 FE 19 8D 8F BE 81 84 98 43 D0 B6 80 14 69 .....C....i
  
```

Figure 3.14: Header of an encrypted file with Meo.

We performed our own analysis on the Meo encryption and decryption process and we determined that the exact same method described in Figure 3.13 is applied in the encryption with the **DES** algorithm. Additionally, even though there are the 4 bytes that store the length of the hashed password we found out that the length remains the same (32 bytes) through our experimentation. That is a logical observation because the output of the MD5 hash algorithm is fixed length.

Finally, we tested the email encryption functionality that Meo offers. Meo encrypts the email body along with the attached files but not the subject line and sends the email to the recipient with an encrypted Meo file that contains both the original email body and attached files (Figures 3.15 & 3.16). Therefore, the same encryption & decryption methods are applied.

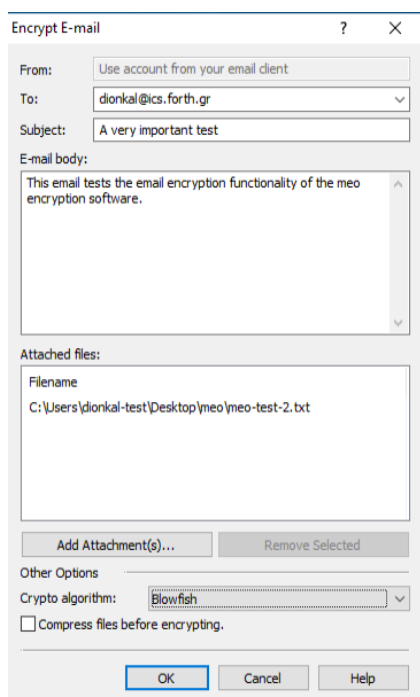


Figure 3.15: Meo email encryption.

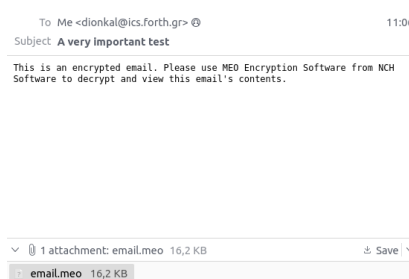


Figure 3.16: Received email encrypted by Meo.

3.1.3.2 Scenario

The technique detailed in the “*Setup*” section can be labeled as “**Guess the Key**” [39] as shown in Figure 2.5. This classification arises from the limited data available through the hashed password, prompting an attempt to brute force the correct password through guessing. Consequently, the success of this method is intricately linked to both the strength of the user’s password and the effectiveness of the wordlist used for the brute force attack.

3.1.3.3 Setup

In order to decrypt files encrypted with Meo, we set up a target environment running in a virtual machine and a forensic workstation to analyze the target environment extraction.

- Target environment
 - Windows 10 x64 Pro version 22H2 on a virtual machine
 - Meo encryption software version 2.17
- Forensic workstation
 - hashcat version 6.2.5 (Released November 2021)
 - Python 3.12.0 (Released October 2023)

We identified all potential files encrypted with 'meo' by collecting a list of files marked with the `.meo` suffix and inspecting file headers (initial 9 bytes) that contain the string "HCN 0.1". Subsequently, we created a script to automate the reversal of the key storage process. We utilized hashcat [30] to attempt password cracking. The outcome of this process varies depending on the strength of the password and the wordlist used. For instance, the password "password123" was easily cracked using the "rockyou" dictionary [64]. On the contrary, when using the same dictionary, hashcat was unable to crack the password "Th1s1s4V3ryL0ngPassword!\$#".

3.1.3.4 Validation

To validate that our findings are forensically sound and that they don't alter evidence in no way, we compared the original plaintext files from the virtual testing environment with the decrypted plaintext files in our forensic workstation.

	MD5
original	c2d9a3bdebd59e4217b844940c76f5
decrypted	c2d9a3bdebd59e4217b844940c76f5
	SHA256
original	de49c390f0aa148059545b1d496cd4887adaf7dffa8490097f39a74365bac9dd
decrypted	de49c390f0aa148059545b1d496cd4887adaf7dffa8490097f39a74365bac9dd

Table 3.7: Process of user password storage

3.2 Memory analysis

Memory analysis in Digital Forensics is the process of examining the volatile memory (RAM) of a computer or digital device to extract and analyze information that is currently stored in the system's active memory. Memory analysis is particularly valuable when dealing with live systems or incidents where time is of the essence. It complements traditional disk-based forensics by providing a real-time snapshot of a device's state.

The significance of memory analysis is emphasized in RFC 3227 [55], particularly in the “2.1 **Order of Volatility**” section. RFC 3227 underlines the importance of addressing volatile memory first in digital forensic investigations to ensure the preservation of critical data that might be lost if not promptly collected. Below we introduce two methods of encryption workarounds with memory analysis, these are (i) BitLocker FVEK extraction, and (ii) KeePass master password recovery.

3.2.1 BitLocker

BitLocker is a full-disk encryption feature included in various versions of Microsoft Windows operating systems. Its primary purpose is to help protect the data on your computer by encrypting the entire hard drive or specific partitions. This encryption ensures that if someone gains physical access to your computer or tries to tamper with your hard drive, they won't be able to access your data without the necessary decryption key. According to Microsoft [46], from Windows 8.1 and onwards, BitLocker Device Encryption is **automatically enabled** on devices that support Modern Standby¹⁴. Furthermore, Microsoft mentions in [46] that it is expected that most devices in the future will automatically enable BitLocker Device Encryption to ensure constant device protection.

3.2.1.1 Architecture

BitLocker utilizes many keys and encryption modes to achieve its encryption. First it encrypts the file data with the **Full Volume Encryption Key (FVEK)**. Next it encrypts the FVEK with the **Volume Master Key (VMK)** and places the encrypted FVEK in the **drive metadata**. Finally, it utilizes various **Key Protectors** to encrypt the VMK and place it in the drive metadata as well. Figure 3.17 illustrates the decryption process of a drive encrypted with BitLocker. Below we explain more about each key component and finally we introduce the encryption workaround method to access the plaintext data.

Full Volume Encryption Key: BitLocker generates the Full Volume Encryption Key (FVEK) and encrypts all the data of a drive or volume using 128 or 256 bit symmetric encryption. According to Microsoft [43], BitLocker Device Encryption supports the following encryption modes:

¹⁴Read more about Modern Standby from Microsoft, <https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/modern-standby>

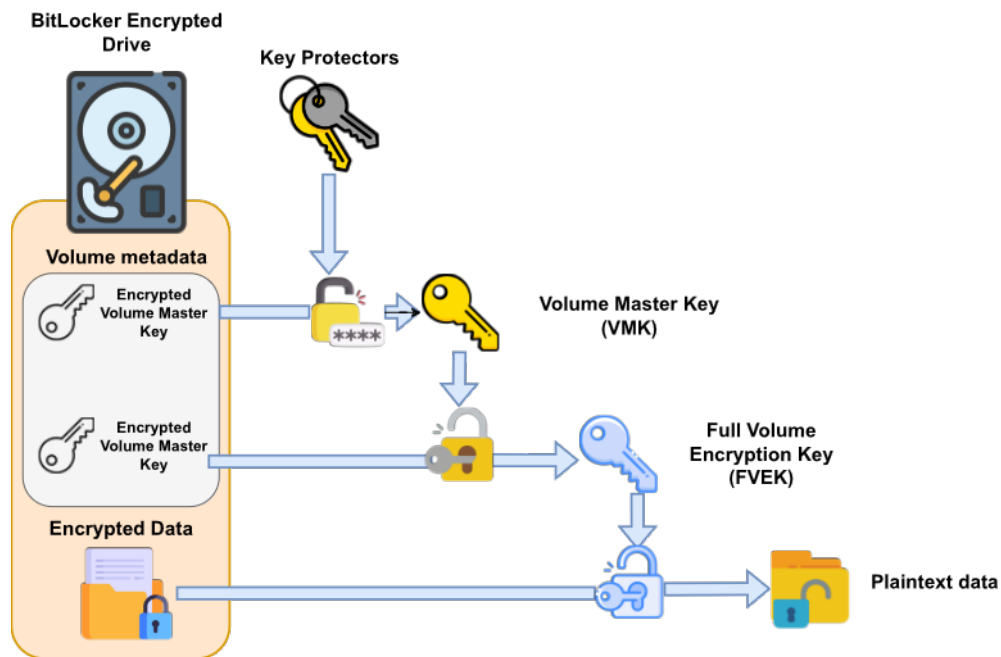


Figure 3.17: BitLocker decryption process

- Windows 8.1 devices
 - AES 128-bit with Diffuser
 - AES 256-bit with Diffuser
 - AES 128-bit (default)
 - AES 256-bit
- Windows 10 or later devices
 - AES-CBC 128-bit
 - AES-CBC 256-bit
 - AES-XTS 128-bit Windows 10 (default)
 - AES-XTS 256-bit Windows 10

Volume Master Key: The purpose of the Volume Master Key (VMK) is to encrypt the Full Volume Encryption Key. It is an intermediate key between the Key Protectors and the FVEK. It utilizes 256-bit symmetric encryption.

Key Protectors: Some of the most popular Key Protector types are listed below, for more information you can refer to [45].

- **Trusted Protection Module:** This is currently the most common and secure method. TPM can either be a separate chip (dTPM) located on

the device's motherboard, or TPM 2.0 enables manufacturers to integrate it into the CPU/motherboard chipset. It is utilized by BitLocker to secure cryptographic keys and to ensure that the underlying operating system has not been tampered with. The TPM stores the key used to decrypt the encrypted VMK from the volume's metadata section. Additionally, TPM can verify the integrity of the system's boot process, ensuring that the system boots from a trusted source and has not been tampered with.

- **User password:** This method automatically decrypts the BitLocker-protected drive using the Windows user password upon the user's login to their account.
- **Recovery password:** BitLocker generates a recovery key for encrypting the VMK. The recovery key serves as a means for the user to unlock a BitLocker-encrypted drive in situations where other methods are unavailable or no longer possible.

Encryption workaround: When a computer is running with a BitLocker-enabled drive, it requires the key to access it. For quick access, it stores the **Full Volume Encryption Key** in the system's RAM. This key is kept within a kernel pool allocation in memory, as illustrated in Figure 3.18. To locate this pool, we perform a memory search technique known as **pool scanning**. During pool scanning, we examine memory pool headers to find pools that match specific criteria. We have the flexibility to search based on any of the fields existing in the `_POOL_HEADER` structure.

Listing 2 offers an overview of the structure. It's worth noting that there's no official documentation available from Microsoft regarding this structure, and there may be potential changes in newer Windows versions. However, the provided fields are sufficient for our purposes as a basis for our search. It's crucial to recognize that pool scanning can potentially yield false positives. To mitigate this, it's advisable to conduct scans with the maximum available parameters.

- **Pool Tag:**
 - **Windows 7:** "FVEc"
 - **Windows 8 & Windows 10 AES-CBC:** "Cngb"
 - **Windows 10 AES-XTS:** "None"
- **Pool size:** This is determined by the length of the key 128-bit or 256-bit and the Windows version
- **Pool type:** Non-paged, due to the fact that the key always remains in memory and cannot be paged

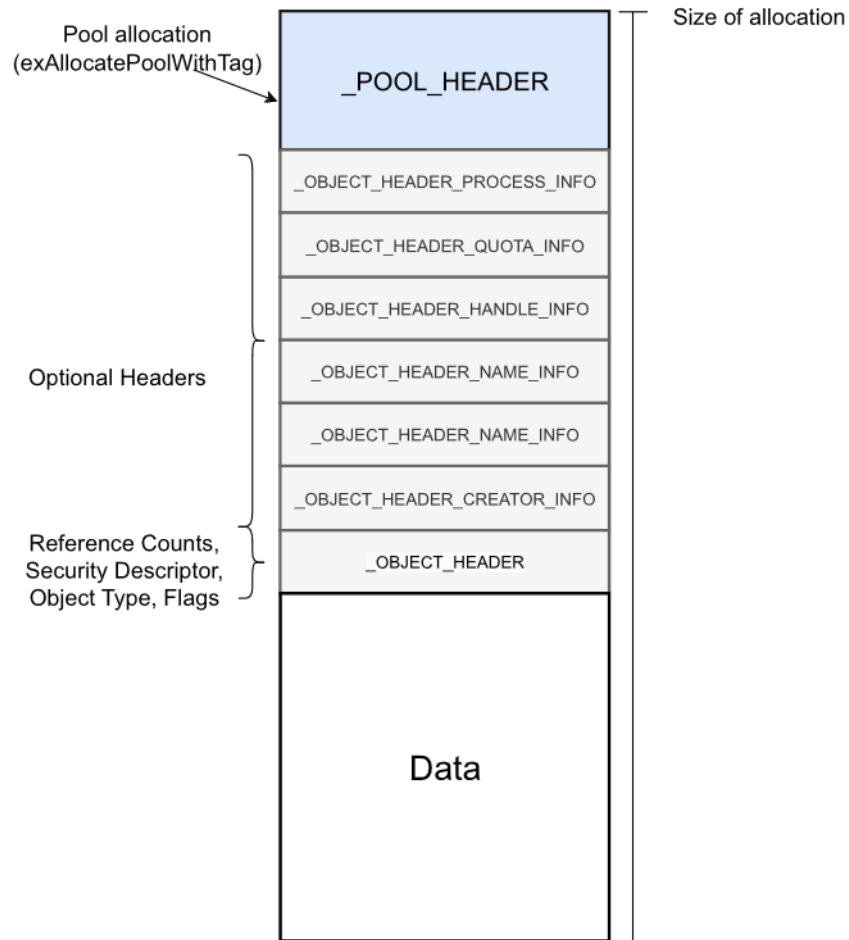


Figure 3.18: Example kernel pool allocation [41].

Listing 2 Simplified `POOL_HEADER` structure in Windows 10 version 1809 as shown in [4]

```

struct POOL_HEADER
{
    char    PreviousSize;
    char    PoolIndex;
    char    BlockSize;
    char    PoolType;
    int     PoolTag;
    Ptr64   ProcessBilled;
}

```

3.2.1.2 Scenario

This method of circumventing encryption falls under the classification of **”Find the Key”** [39], as it involves the extraction of the FVEK necessary to unlock a BitLocker-encrypted drive. Furthermore, we firmly believe that law enforcement professionals will encounter a growing number of devices secured with BitLocker. Microsoft’s statement regarding its increasing prevalence [46], along with the default enablement on many newer devices, underscores its role in providing seamless protection to users.

3.2.1.3 Setup

To test the encryption workaround we used the following setup:

1. Target computer: Windows 10 Pro 64-bit version 15.19041. The operating system was running natively on computer and not on a Virtual machine.
2. Forensic workstation: Linux Ubuntu 22.04.3 LTS x86_64

We activated BitLocker encryption on the target computer and captured its active memory using FTK Imager (version 4.7.1). We then transferred the memory dump to our forensic workstation for analysis. Our choice for memory analysis was **”Volatility”** (version 2.6.1), a widely recognized open-source tool known for its ability to handle various memory file formats and adapt to changing operating system structures. This flexibility streamlined our focus on image analysis.

Moreover, we extended Volatility’s capabilities by integrating a third-party plugin (accessible at [6]) to aid in memory examination and the extraction of the BitLocker FVEK, as explained in the pool scanning methods outlined in the **”Architecture”**.

In the following sections, we outline the specific steps taken for each stage of the process, with **”Acquisition”** providing details on how we obtained both the system’s memory and drive.

Acquisition: Our methodology began with the initial step of acquiring evidence, which involved capturing the active memory of the running target system. Subsequently, following the system’s shutdown, we proceeded with a physical acquisition of the encrypted BitLocker drive from the target system.

- **Memory:** It is required for the target system to be running and logged in to the windows user account, then we connected a USB Drive that had the FTK Imager 4.7.1 installed and run it the target system as seen in Figure 3.19. The size of the RAM of the target computer was 32GB.
- **Encrypted Drive:** To acquire the drive we powered off the target system. Next, we inserted a live USB running Kali Linux and booted in forensics mode. In forensic mode, Kali Linux is launched directly from the USB, and

it refrains from automatically mounting any drives. Offensive Security’s documentation regarding the forensics mode, as mentioned in [26], emphasizes that in this mode, absolutely **nothing** should happen to **any** media without **direct user action**. To capture the drive we connected an external disk large enough to accommodate the size of the BitLocker drive, then run the guymager tool [29] that comes with Kali Linux to acquire the drive.

For more details about the acquisition process using guymager, please consult Figures A.2 & A.3. The size of the NVMe in the target system that was acquired was 512GB.

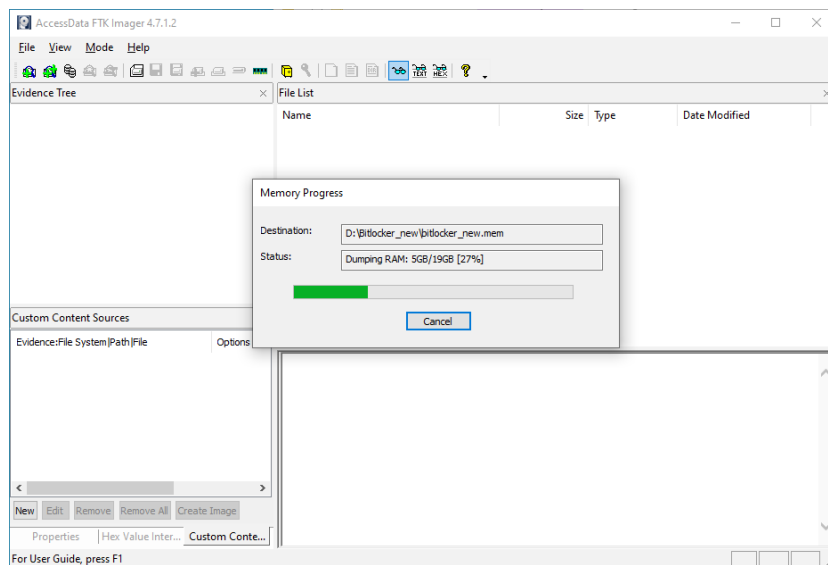


Figure 3.19: Live memory acquisition with FTK Imager

Memory analysis: To unlock the BitLocker drive we first needed to extract the FVEK from the acquired memory dump.

- **Determining memory profile:** First we needed to find the memory profile of the extracted memory dump:

```
$ python2 vol.py -f <path_to_bitlocker_memdump>
imageinfo
```

According to Figure 3.20 we can see that the suggested memory profile for our memory dump is the profile "Win10x64_19041". Now we are ready to run the BitLocker plugin¹⁵ and extract the BitLocker FVEK from the memory dump.

¹⁵Volatility-BitLocker, <https://github.com/breppo/Volatility-BitLocker>

```

dionkal /master - 1 Documents > volatility 1 python2 vol.py -f ~/Desktop/CYBERSPACE/Artifacts/Bitlocker/bitlocker_mendump.mem inagetinfo
Volatility Foundation Volatility Framework 2.6.1
INFO : volatility.debug : Determining profile based on KDBG search...
Suggested Profile(s) : Win10x64_19041
AS Layer1 : SkipDuplicatesAMD64PagedMemory (Kernel AS)
AS Layer2 : FileAddressSpace (/home/dionkal/Desktop/CYBERSPACE/Artifacts/Bitlocker/bitlocker_mendump.mem)
PAE type : No PAE
DTB : 0x1ad002L
KDBG : 0xf80228406b20L
Number of Processors : 12
Image Type (Service Pack) : 0
KPCR for CPU 0 : 0xfffff80223da2000L
KPCR for CPU 1 : 0xfffff80132fca000L
KPCR for CPU 2 : 0xfffff801331ec000L
KPCR for CPU 3 : 0xfffff80133680000L
KPCR for CPU 4 : 0xfffff80133740000L
KPCR for CPU 5 : 0xfffff8013338e000L
KPCR for CPU 6 : 0xfffff80133882000L
KPCR for CPU 7 : 0xfffff80133940000L
KPCR for CPU 8 : 0xfffff80133a00000L
KPCR for CPU 9 : 0xfffff80133ac0000L
KPCR for CPU 10 : 0xfffff80133ba0000L
KPCR for CPU 11 : 0xfffff80133be4000L
KUSER_SHARED_DATA : 0xfffff78000000000L
Image date and time : 2023-06-29 11:14:59 UTC+0000
Image local date and time : 2023-06-29 04:14:59 -0700

```

Figure 3.20: Memory dump information extracted by the Volatility framework.

- **Extracting the FVEK:** The dislocker switch (`--dislocker`) instructs the BitLocker plugin to dump all the candidate FVEKs in files compatible with the dislocker program. Each key will be stored in a separate file inside the directory we specified. In our case inside the `fvek` directory.

```

$ python2 vol.py -f <path_to_bitlocker_mmedump>
--profile Win10x64_19041 bitlocker --dislocker
fvek/

```

Mount the drive:

The extracted BitLocker encrypted drive is in **Expert Witness Format (ewf)**. To mount the ewf extraction we used the `ewfmount` version 20140807. It can be found in `libewf` [40] which contains a collection of ewf related tools.

```

$ mkdir /mnt/ewf_mount
$ sudo ewfmount bitlocker_drive.E01 /mnt/ewf_mount

```

Next, we examined the mounted ewf image to identify all the partitions as seen in figure 3.21. The main Windows partition that we are interested in is **ewf1p3** as it is the biggest by far and its type is labeled as “Microsoft basic data”. Dislocker has the option to mount a selected partition by specifying the offset in **bytes**, as we are interested in the third partition `ewf1p3` which starts in sector 239616 and the dislocker output in the second line states that are 512 bytes per sector that means that our offset to mount the `ewf1p3` partition will be $239616 \times 512 = 122683392$.

After we figured the offset of the partition, we created two more mount points:

1. dislocker file:

```

$ mkdir /mnt/dislocker

```

```
dionkal ~ > dionkal > AcquisitionHDD > Bitlocker_new 1 sudo fdisk -l /mnt/ewf_mount/ewf1
Disk /mnt/ewf_mount/ewf1: 476,94 GiB, 512110190592 bytes, 1000215216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 87CFBC91-5EC5-4DA5-A9B1-C10ED43754D8
```

Device	Start	End	Sectors	Size	Type
/mnt/ewf_mount/ewf1p1	2048	206847	204800	100M	EFI System
/mnt/ewf_mount/ewf1p2	206848	239615	32768	16M	Microsoft reserved
/mnt/ewf_mount/ewf1p3	239616	999141924	998902309	476,3G	Microsoft basic data
/mnt/ewf_mount/ewf1p4	999143424	1000212479	1069056	522M	Windows recovery environment

Figure 3.21: List partitions of the Windows drive.

2. final folder that will mount the dislocker file:

```
$ mkdir /mnt/disk
```

Finally, we run the dislocker program found at [1] to unlock the BitLocker encrypted drive and mount it to our system.

```
$sudo dislocker -k <path_to_fvek_dump> /mnt/ewf_mount/ewf1
-O 122683392 /mnt/dislocker
```

```
$sudo mount /mnt/dislocker/dislocker-file /mnt/disk
```

Dislocker typically attempts to mount the drive in read-write mode by default. However, in the case of an EWF file extraction, mounting with write permissions is not possible. This limitation is the reason behind the error shown in Figure 3.22. This behavior is actually beneficial, as it ensures that the drive is mounted in read-only mode, thereby preventing any unintended alterations to electronic evidence.

```
dionkal ~ > sudo dislocker -k Desktop/fvek_dump/0xe40230f30890-Dislocker.fvek /mnt/ewf_mount/ewf1 -O 1
22683392 /mnt/dislocker/ && sudo mount /mnt/dislocker/dislocker-file /mnt/disk/
Error opening '/dev/loop31' read-write
The disk contains an unclean file system (0, 0).
Metadata kept in Windows cache, refused to mount.
Falling back to read-only mount because the NTFS partition is in an
unsafe state. Please resume and shutdown Windows fully (no hibernation
or fast restarting.)
Could not mount read-write, trying read-only
```

Figure 3.22: Unlocking BitLocker drive and mounting it at /mnt/disk.

We have verified the successful mounting of the Windows partition at our designated mount point, /mnt/disk. This confirmation allows us to access all files within the Windows filesystem, as depicted in Figure 3.23.

```
dionkal / >> mnt ls drive
'SRecycle Bin' intel 'Program Files' 'System Volume Information'
'Documents and Settings' pagefile.sys 'Program Files (x86)' Users
DumpStack.log.tmp PerfLogs Recovery windows
hiberfil.sys ProgramData swapfile.sys
```

Figure 3.23: Contents of the unlocked BitLocker drive.

3.2.1.4 Validation

In order to ensure the integrity of the process outlined in the "Setup" section, confirming that it did not modify the memory dump, we conducted a comparison of the MD5 and SHA256 hashes both before and after analyzing the memory dump. The results displayed in Table 3.8 indicate that the memory dump remains unaltered, as neither the MD5 nor the SHA256 values exhibit any changes following our analysis.

	MD5
before	2e9203dda24978c586e2f3f0b35771
after	2e9203dda24978c586e2f3f0b35771
	SHA256
before	e697f4006b0e4e636dbbce3250bbbc8e4c76df78ed97b35eae7045d2c4270522
after	e697f4006b0e4e636dbbce3250bbbc8e4c76df78ed97b35eae7045d2c4270522

Table 3.8: MD5 & SHA256 hashes of the memory dump

3.2.2 KeePass

KeePass is an open-source password manager and secure digital vault that enables users to store their passwords and other sensitive information in a secure manner. It's a versatile solution that's available on multiple platforms, including Windows, Linux, and MacOS, ensuring accessibility across various operating systems. KeePass offers offline storage for user secrets, keeping your data secure without relying on cloud-based functionality. Additionally, it provides a highly customizable password generator, enabling you to create strong and random passwords with ease.

3.2.2.1 Architecture

In May 2023, a new vulnerability was announced [67] that enabled an attacker to retrieve the KeePass master password by examining the system memory. The developer of KeePass subsequently confirmed this vulnerability [67] and was assigned the CVE-2023-32784 and a severity score of 7.5 (High) by the NIST National Vulnerability Database [51]. This vulnerability affected all versions of KeePass 2 released up to that point, the most recent being 2.53.1. KeePass 2 employs a custom-developed text box for password input, known as "*SecureTextBoxEx*". This text box is not only used for entering the master password but also in other parts of KeePass, such as password edit boxes. Consequently, this attack can also be used to recover the contents of these password edit boxes. The underlying flaw exploited in this scenario is that, for each character typed, a residual string is generated in memory. Due to the way .NET functions, eliminating these strings once they are created is a formidable challenge. For example, if you were to type "Password," it would result in the following leftover strings: ●a, ●●s, ●●●s, ●●●●w, ●●●●●o, ●●●●●●r, ●●●●●●●d. This characteristic permits partial recovery of the master password. The initial character of the password remains unrecoverable, while options exist for recovering the second character and all subsequent characters accurately. It's crucial to note that the only requirement for the exploit to succeed is that the target user must have manually typed the master password to unlock the KeePass database. Copy-pasting the password will not leave the password leftover strings in memory. Importantly, KeePass does not need to be actively running for this attack to be effective.

Mitigation KeePass version 2.54 released in June 2023 [38], approximately one month after the flaw became public. This version addressed the issue, and as of version 2.54 and beyond, it is no longer possible to recover parts of the master password from memory.

3.2.2.2 Scenario

This exploit falls within both the "*Exploit a flaw*" and "*Guess the key*" categories in the encryption workaround taxonomy classification introduced by Schneier et al. [39]. It is a disclosed exploit and assigned a CVE by NIST NVD [51], hence the

”*Exploit the flaw*” classification. However, it’s essential to understand that this exploit does not provide direct access to the password vaults by circumventing the encryption and authentication mechanisms. Instead, it enables an attacker to recover the partial master password, excluding only the first two characters. Consequently, the attacker must engage in a guessing process, as detailed in the ”*Setup*” section, which falls under the ”*Guess the Key*” category.

Forensic value Gaining access into a KeePass vault is of paramount importance in the context of a forensic investigation, as it could provide valuable evidence pertinent to a case. Within a KeePass vault, a wide array of information can be stored, encompassing user credentials for online accounts and various confidential data, including, but not restricted to, potentially incriminating communications and illicit media content that a suspect might seek to conceal from law enforcement investigators.

Exploit longevity As mentioned in the *Architecture* section a fix for the exploit has been released one month after the disclosure. Suspects who have upgraded to a newer version will not be affected by the exploit thus making the decryption impossible. Although the impact of a high severity flaw might be of significant value for forensic investigators, there is a limited time frame that the flaw can be leveraged.

Vulnerability Disclosure The longevity of an exploit can be extended if the vulnerability is not disclosed to the software vendor. However, this approach may potentially jeopardize the security and privacy of regular users who rely on the affected software. Some countries like the US as documented in [32] and UK as outlined in [27] have implemented a **Vulnerability Equity Process (VEP)**. This program involves the assessment of undisclosed vulnerabilities that have been discovered by relevant government agencies. The assessment is based on the vulnerabilities’ utility and potential impact on the public. In contrast, Europol as referenced in [11] has emphasized that weakening encryption through the introduction of backdoors or not disclosing discovered vulnerabilities could ultimately facilitate the work of malicious actors. It’s important to note that the primary aim of this thesis is not to make assertions about disclosure policies. Instead, the thesis is intended to present and evaluate methods that Law Enforcement Agencies (LEAs) can employ to access decrypted data. In the case of vulnerability exploitation, the only assertion we can make is that its longevity is typically limited, necessitating continuous research for new vulnerabilities.

3.2.2.3 Setup

To test the exploit, we set up a new virtual machine running Windows and installed KeePass. Next, we created a new vault within KeePass and set the **master password** as ”**password123**”. In order to capture a memory dump of the entire

RAM on the target system we installed the tool FTK Imager. This tool is capable of performing a whole RAM capture and can also acquire the system's page file.

Below are the specific version details of all the software and tools we used.

- Target: Windows 10 Pro x64 running inside VirtualBox
- Software: KeePass 2
 - version: 2.53.1
 - Vault master password: *password123*
- Acquisition tool: FTK imager 4.7.1
 - *memdump.mem*: Memory dump of the whole RAM
 - *pagefile.sys*: Pagefile of the operating system
- Windows Task Manager:
 - *Keepass.DMP*: Dump of the KeePass process memory. This was done by navigating to the Windows process list, right clicking the KeePass process and selecting the option "Dump process" [Figure 3.24]

Analysis System We set up another environment that will act as the investigator's computer that will perform the decryption process.

- Investigator's VM: Kali Linux 64bit 2023
- dotnet environment
- CVE-2023-32784 proof of concept script [68]

Exploitation After we have extracted the memory dump and setup dotnet on our system we are able to run the exploit:

```
$ cd <path to exploit>
$ dotnet run <path to dump>
```

In both cases, memory (see Figure 3.25) & process dump (see Figure 3.26), we were able to correctly recover almost all the characters of the password, with an exception of the two first characters. The first character cannot be recovered, and for the second character we have a list of candidate characters. We observed that the process dump has less possible options for the second character, possibly because of the size of the dump. In our experiment the process dump was 286.7 MB whereas the memory dump was 9.1 GB. Therefore it is possible that the dump

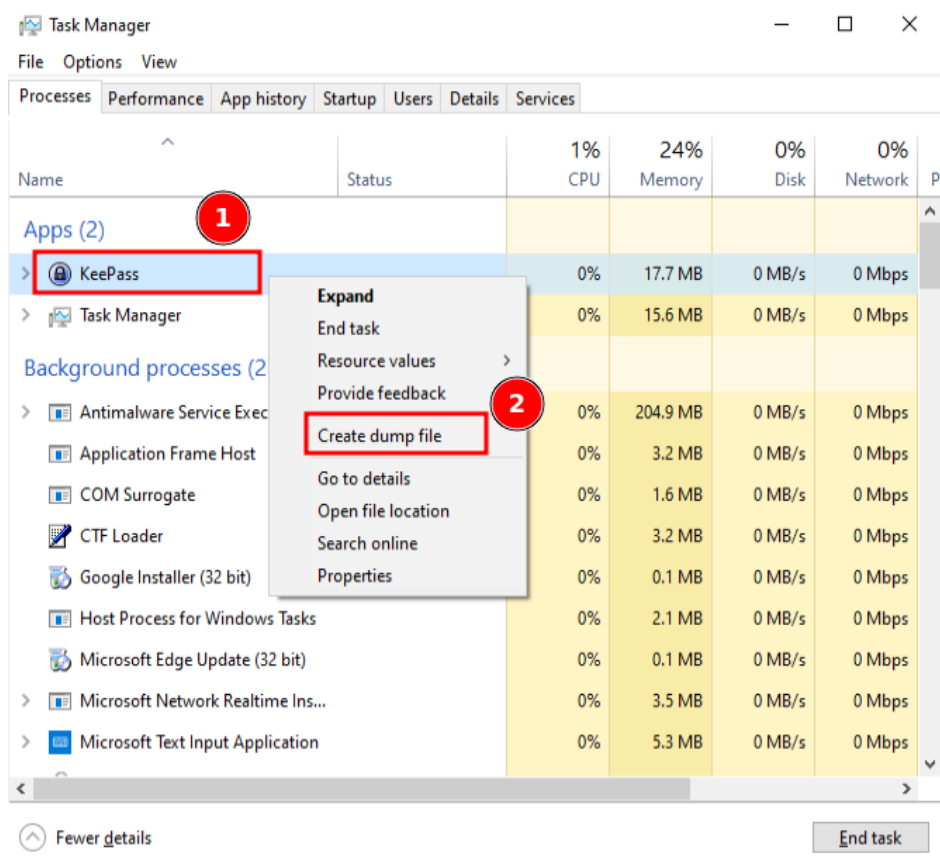


Figure 3.24: Dump process memory from Windows task manager

of the whole memory might include more false positives. It is important to note thought, that in both cases the correct character 'a' is included in the list.

However, we did not succeed into recovering the password while using the system's page file. We attribute this to the fact that the system was not used extensively for a long time during our testing. Therefore the system did not need to write the KeePass master password to the page file. Nonetheless, we firmly believe that in a real life situation we would be able to recover the password from the page file, since it has the same raw format as the memory and process dumps, provided that the computer had previously paged the KeePass memory.

```

Password candidates (character positions):
Unknown characters are displayed as "●"
1.: ●
2.: }, g, `, A, e, c, 2, i, u, q, s, o, S, K, [, Y, {, U, k, 5, h, t, l, T,
#, W, ', $, ", E, , _*, w, |, y, &, p, <, !, J, %, -, ,, ., :, ^, B, G, C, Q,
f, d, 1, r, v, F, ?, 7, I, ), /, >, 4, +, a, 0, H, m, @, R, 9, x, =, Z, b, D,
3.: s,
4.: s,
5.: w,
6.: o,
7.: r,
8.: d,
9.: 1,
10.: 2,
11.: 3,
Combined: ●[, g, `, A, e, c, 2, i, u, q, s, o, S, K, [, Y, {, U, k, 5, h, t, l,
T, #, W, ', $, ", E, , _*, w, |, y, &, p, <, !, J, %, -, ,, ., :, ^, B, G, C,
Q, f, d, 1, r, v, F, ?, 7, I, ), /, >, 4, +, a, 0, H, m, @, R, 9, x, =, Z, b,
D}ssword123.

```

Figure 3.25: Output of the PoC using the memory dump

```

Password candidates (character positions):
Unknown characters are displayed as "●"
1.: ●
2.: a, A, B, G, I, ], ^, F, 5, 8, 9, ., ), Y, 4, ',
3.: s,
4.: s,
5.: w,
6.: o,
7.: r,
8.: d,
9.: 1,
10.: 2,
11.: 3,
Combined: ●[a, A, B, G, I, ], ^, F, 5, 8, 9, ., ), Y, 4, '}ssword123

```

Figure 3.26: Output of the PoC using the process dump

Finally, to retrieve the whole password we will have to perform educated guesses on the first two characters. For the second character we have **a) 77** (memory dump) and **b) 16** (process dump) candidates. For the first character we do not have any information other than that it can be any character out of the **256**. So we would have to do (a) $77 \times 77 = 19.712$ or (b) $16 \times 256 = 4096$ guesses at most. We could also argue that some of the 256 characters are not readable and cannot be typed by a keyboard to further limit the guesses we have to make.

Further more, we might be able to deduce the whole password by observing the rest of the characters that we know. It is known from the early days by security researchers that users usually pick predictable passwords [48] such as words that are easier to remember but also easier to guess if you know part of the password.

For example, in our case, we can readily deduce that the first and second characters are “p” and “a”, based on our knowledge that the master password, from the third to the last character, is “ssword123”. Likewise, an experienced

investigator possessing insights into the suspect's character may be able to infer the rest of the suspect's Keeppass master password.

3.2.2.4 Validation

In the domain of digital forensics, it is crucial for investigators to validate their tools and verify the integrity of electronic evidence to ensure its admissibility in a court of law. One significant concern is the potential alteration of electronic evidence during the investigative process. In this case, that we're dealing with an exploit that opens digital dumps as read-only files, it remains paramount to verify that this process does not in any way modify the contents of these dumps. This is especially important because both of these digital dumps are likely to be treated as electronic evidence in a legal context.

In Table 3.9 we observe the md5 & sha256 hash of both the process and memory dumps from before and after running the exploit remain unchanged. Therefore, we conclude that the exploitation process is forensically sound.

Process dump file	
MD5	
before	39228a432e0d5165f85396eced6d1bbf
after	39228a432e0d5165f85396eced6d1bbf
SHA256	
before	61e95a8271c36eba7bf9f6cd7a8a5f1d36ff1e2edeb8ea1e1aea50e13f455ddd
after	61e95a8271c36eba7bf9f6cd7a8a5f1d36ff1e2edeb8ea1e1aea50e13f455ddd
Memory dump file	
MD5	
before	d8a8bde6b9b0aa16f61c320e2832698a
after	d8a8bde6b9b0aa16f61c320e2832698a
SHA256	
before	11f97daec7dd62b12f16e71c8f69a6b352e969df15f9520432cd7696f63e4646
after	11f97daec7dd62b12f16e71c8f69a6b352e969df15f9520432cd7696f63e4646

Table 3.9: MD5 & SHA256 hashes of the process & memory dump files before and after running the exploit

Chapter 4

Implementation

In this chapter we present our contribution, **AWLPS**, a platform designed to automatically execute the encryption workarounds detailed in the “Methodology” chapter. First, we examine related open source security tools and extract the foundational characteristics that a security tool of this scope must have. Then, in the “Architecture” section we delve deeper in our design philosophy for the tool following the basic guidelines we established in the “Related Tools & Principles” section. Finally, we present our implementation of the AWLPS platform along with a Graphical User Interface and the volatility integration features.

4.1 Related Tools & Principles

We examined the following open-source security tools favored by security professionals to understand which features make a security tool useful:

1. **Autopsy**¹

Autopsy is an easy-to-use platform for Digital Forensics and a graphical interface to the Sleuthkit² and other digital forensics tools. It is used by law enforcement, military, and corporate examiners to investigate what happened on a computer.

2. **Metasploit Framework**³

Metasploit Framework is an open-source penetration testing framework designed for developing, testing, and executing exploit code against a remote target machine. It provides a comprehensive set of tools for penetration testing, vulnerability assessment, and security research.

¹See www.autopsy.com/

²See <https://sleuthkit.org/sleuthkit/>

³See www.metasploit.com/

3. Volatility 3⁴

Volatility is the world's most widely used framework for extracting digital artifacts from volatile memory (RAM) samples. The extraction techniques are performed completely independent of the system being investigated but offer visibility into the runtime state of the system. The framework is intended to introduce people to the techniques and complexities associated with extracting digital artifacts from volatile memory samples and provide a platform for further work into this exciting area of research.

4.1.1 Extensibility

The open-source software referenced in the preceding section are designed to be expanded by external parties. This indicates that if any component becomes obsolete or if new security features are required, they can be seamlessly integrated thanks to the plugin architecture allowing the tool to be useful and relevant for the future. These tools establish an abstract entity that outlines the requirements for all plugins, allowing plugin authors to create new plugins in accordance with these specified guidelines.

Autopsy offers the following kinds of modules ⁵ that third parties can implement to extend the usability of the tool.

- **Ingest Modules**

These modules are run when a new data source (such as a disk image) is added to a case.

- **Report Modules**

These modules are run after the user has reviewed results and tagged files. Their intention is to create an output report of the results, but they can also be used to perform analysis.

- **Content Viewers**

These modules are graphical and focus on displaying a specific file in some unique way. An example of a viewer module is to view the file in hexadecimal, extract the strings from the file, and view images and movies.

- **Result Viewers**

These modules show information about a set of files. These modules are in the upper right of the interface. The platform comes with viewers to view the set of files in a table and thumbnails.

⁴See www.volatilityfoundation.org/releases-vol3

⁵Refer to https://www.sleuthkit.org/autopsy/docs/api-docs/4.1/platform_page.html for more details about Autopsy modules

Metasploit also offers the ability to write custom modules⁶. The **post-exploitation** modules are responsible for searching through a target host that is compromised. The categories of the post-exploitation modules are explained in Table 4.1. The module types related to our objective are the “gather/credentials” and “gather/forensics” that already have numerous modules that gather credentials⁷.

Category	Description
gather	Modules that involve data gathering/collecting/enumeration.
gather/credentials	Modules that steal credentials.
gather/forensics	Modules that involve forensics data gathering.
manage	Modules that modifies/operates/manipulates something on the system. Session management related tasks such as migration, injection also go here.
recon	Modules that will help you learn more about the system in terms of reconnaissance, but not about data stealing. Understand this is not the same as “gather” type modules.
wlan	Modules that are for WLAN related tasks.
escalate	This is deprecated, but the modules remain there due to popularity. This used to be the place for privilege escalation modules. All privilege escalation modules are no longer considered as post modules, they’re now exploits.
capture	Modules that involve monitoring something for data collection. For example: key logging.

Table 4.1: Categories of post-exploitation modules in metasploit framework.

⁶Refer to <https://docs.metasploit.com/docs/development/developing-modules/guides/how-to-get-started-with-writing-a-post-module.html> to see documentation on how to write modules.

⁷For a complete list of modules refer to docs.metasploit.com/docs/modules.html, there are about 100 modules related to encryption and credentials, alternatively start an msfconsole session and write the following command “search credentials”.

Volatility 3, offers relatively few modules related to encryption and credential gathering. The `hashdump`⁸ and `cachedump`⁹ modules are responsible for dumping windows user password hashes and LSA¹⁰ secrets respectively. Volatility 2 (which is now outdated) also had a module to dump the Truecrypt device encryption masterkey and passphrase and extracting the BitLocker FVEK seen in Chapter 3.2.1.

In a similar fashion, we structured our tool to be **adaptable**, considering that encryption workarounds may become obsolete or new ones may emerge.

Moreover, we anticipate that this modularity will contribute to the enhancement of **collaboration** among Law Enforcement Agencies (LEAs), aligning with a primary goal of the European Union’s Internal Security Fund program [37]. This design empowers LEAs with advanced technical capabilities to seamlessly incorporate and distribute new encryption workarounds to those with limited technical resources.

4.1.2 Forensic Soundness

As explained in [42], **forensic soundness** is used to describe the forensic process as a whole with two clear objectives: (i) the acquisition and subsequent analysis of electronic data has been undertaken with all due regard to preserving the data in the state in which it was first discovered, and (ii) the forensic process does not in any way diminish the evidentiary value¹¹ of the electronic data through technical, procedural or interpretive errors.

Autopsy ensures forensic soundness through its “verify image integrity” process¹², which validates the forensic image and files generated by Autopsy using MD5 hashes.

4.1.3 Operating System Interoperability

The versatility of these tools to operate across a wide range of desktop operating systems offers users the flexibility they require, tailored to their specific demands and usage scenarios. For instance, one forensic laboratory may use Windows workstations, while another may opt for Linux. A proficient investigator can seamlessly

⁸See <https://volatility3.readthedocs.io/en/stable/volatility3.plugins.windows.hashdump.html>

⁹See <https://volatility3.readthedocs.io/en/stable/volatility3.plugins.windows.cachedump.html>

¹⁰Read more about LSA here: <https://learn.microsoft.com/en-us/windows-server/security/windows-authentication/windows-authentication-architecture>

¹¹See examples of forensic mistakes during the acquisition phase <https://www.flashbackdata.com/top-3-mistakes-made-scene-digital-evidence/>

¹²Autopsy Data Source Integrity Module, https://sleuthkit.org/autopsy/docs/user-docs/4.19.3/data_source_integrity_page.html

transition from one operating system to another as required, eliminating the necessity to learn a new security tool each time. All the tools mentioned at the start of this section can be run from both Windows and Linux¹³.

4.1.4 Shortfalls with regards to our Encryption

Autopsy, Metasploit and Volatility are great tools for their intended purpose but fall short when it comes to encryption workarounds.

- Autopsy lacks plugins for circumventing encryption¹⁴; instead, its available plugins are primarily focused on analyzing artifacts with the assumption that they are not encrypted. The only module related to encryption is the encryption detection plugin which uses entropy to detect if a file might be encrypted and does not perform any kind of decryption¹⁵.
- Metasploit is designed as an offensive tool primarily utilized by penetration testers and adversaries. While it does offer post-exploitation modules, such as `post/firefox/gather/passwords`¹⁶, valuable for our purposes (as detailed in Chapter 3), their execution necessitates a pre-established session [49], achievable by obtaining access to a privilege shell. In the context of a digital forensics investigation, analysts typically receive copies of disks or memory from seized devices for analysis. Metasploit, however, relies on the target system being operational, directly accessing it through exploits that could potentially modify its contents, thereby posing a risk to evidence preservation, a critical requirement. Additionally, it lacks a mechanism for image verification through hash calculations before and after module execution.
- Volatility 3¹⁷ is one of the most well known tools among digital forensics practitioners for memory analysis. The “Memory Layer” feature removes the need to know the intricacies of physical memory so that the user is only dealing with virtual addresses. Additionally, the memory layout often changes even between versions of the same operating system, Volatility 3 offers “Symbol Tables” which solve this issue by providing a map of the

¹³Autopsy in Windows operates as an installed program with a graphical interface, in Linux Autopsy sets up a local server to invoke tools from TheSleuthKit and display it in browser.

¹⁴This Github repository contains modules for Autopsy. At the time of writing this thesis, it does not have anything related to decryption or credential gathering, github.com/sleuthkit/autopsy_addon_modules

¹⁵Official documentation of how the encryption detection plugin works, nothing related to decryption, www.sleuthkit.org/autopsy/docs/user-docs/4.5.0/encryption_page.html

¹⁶See docs.metasploit.com/docs/pentesting/metasploit-guide-post-gather-modules.html

¹⁷For a complete list of all the Volatility 3 plugins that ship with tool refer to [volatility3.readthedocs.io/en/stable/volatility3.plugins.html](https://readthedocs.io/en/stable/volatility3.plugins.html)

memory layout and the location of memory objects for a specific operating system version.

Even though, Volatility 3 is a powerful tool in the digital forensic investigators arsenal, its scope is limited to forensic examination of memory images, making it incapable of retrieving data exclusively stored on the disk of an acquired system. That is why we decided to integrate Volatility 3 in our tool to handle the memory analysis aspect as explained in Section 4.2.3.

4.2 Architecture

Due to the shortfalls we mentioned in the previous section we designed an architecture that will have the aforementioned benefits in mind but also address the shortfalls to the specific use of encryption workarounds in digital forensics.

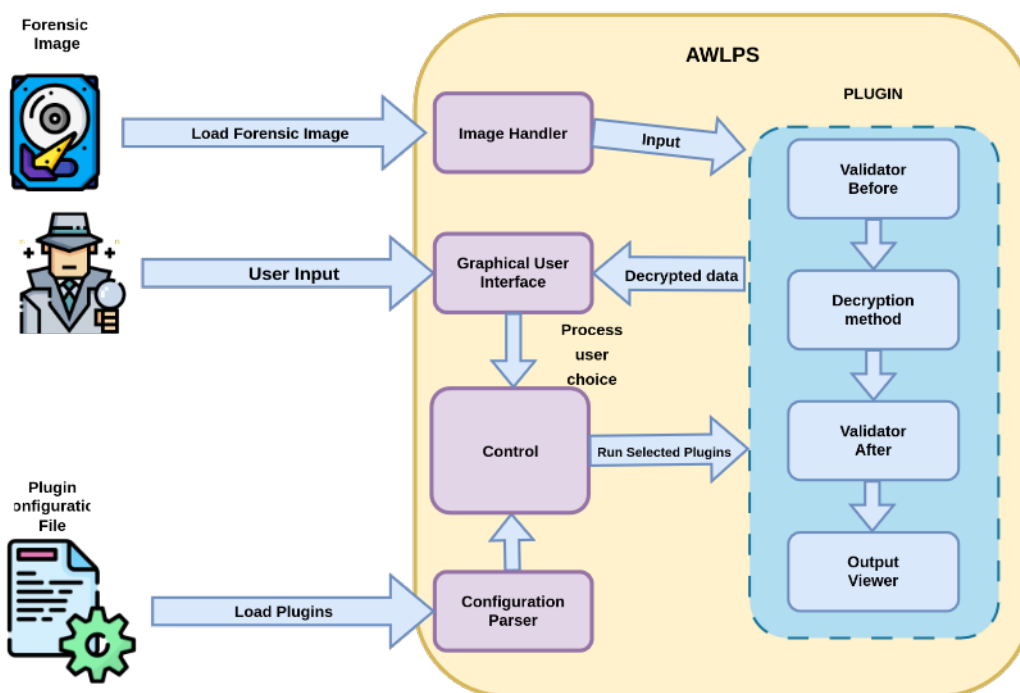


Figure 4.1: AWLPS architecture diagram.

Figure 4.1 illustrates how the components of the AWLPS tool interact with each other.

- **Configuration Parser:**

Reads the configuration file and passes the configuration options such as available plugins and logging to the Control component.

- **Control:**

The control component is responsible for orchestrating the whole operational process of the tool. It translates user input through the GUI to actions inside the tool. It implements the configuration instructions supplied by the configuration parser such as dynamically loading the decryption modules and also instructing which modules to run.

- **Plugin:**

The plugin component is responsible for handling all the operations regarding an encryption workaround module. This includes the decryption operation, the validation before and after the decryption to guarantee forensic soundness and the output viewer that is responsible for displaying the results back to the user.

- **Image Handler:**

This component is responsible for importing, managing and cleaning the input of the tool. Input can be in various forms such as logical filesystems, memory dumps and even EWF files that require specialized handling.

- **Graphical User Interface (GUI):**

Enables the user to easily interact with the tool and instruct the Control component.

Furthermore, the implementation of the architecture uses Python 3 and the chosen dependencies have been selected because they are available on every operating system, thereby ensuring **operating system interoperability**.

4.2.1 Plugin Architecture

AWLPS adopts a plugin architecture, primarily because it enables **extensible** features allowing for the seamless addition of new encryption workarounds. This architectural choice aligns with the trend in security tools, as evidenced by the tools discussed in 4.1 **Related tools & Principles**, all of which adhere to the plugin architecture paradigm.

In AWLPS the plugin and configuration file components are responsible for adding the extendable functionality by abstracting and generalizing the operations of the decryption task to accommodate **extensibility**.

- **Plugin:**

The plugin is designed to encapsulate the fundamental core functionality required for an encryption workaround module. To implement this in Python 3, we utilized the Abstract Base Class¹⁸. To develop an encryption workaround

¹⁸<https://docs.python.org/3/library/abc.html>

plugin the developer must define a class that **inherits** from the abstract Plugin class. To create an encryption workaround plugin, the developer needs to define a class that **extends** the abstract Plugin class. The Plugin class is essentially a contract that the developer should adhere to by implementing the definitions of abstract methods in the inheriting class. These methods are listed below:

- **decrypt**

This is the core function of the plugin that will handle the analysis and decryption process for the plugin.

- **render**

This method is responsible for consolidating all the data of the module such as input, output as well as metadata generated by the validation mechanisms for display back in the GUI.

- **validators**

Is a dictionary that registers all the validation functions that a module will run to validate its input before and after running.

Annex B.2 further elaborates on the methodology to develop an example plugin for the tool and how to override each of the above mentioned methods.

- **Configuration file:** Apart from the plugin itself, AWLPS reads a configuration file at start up in order to obtain information related to the available encryption workaround plugins and other configurable information.

Listing 3 illustrates an example of a configuration file for the tool. The information it includes is the following:

- Tool configuration options such as the logging level.
- The available encryption workaround modules with their descriptions and the location of the files.
- The location and description of dictionaries that are used inside the tool to perform dictionary attacks and crack hashes.

Table 4.2 contains all the fields that a module entry should have along with a description of how each field is utilized within the tool.

Listing 3 Example of AWLPS configuration file format.

```

{
  "logging_enabled": true,
  "log_level": "DEBUG",
  "modules": {
    "module_1" : {
      "id": "example_id",
      "className": "example_class_name",
      "name": "Example display name",
      "developer": "Example display developer",
      "category": "Example display category",
      "enabled": true,
      "platform": "MEMORY"
    },
    "...",
    "module_n": {}
  },
  "dictionaries": [
    {
      "name": "relative path to dictionary file
        /dictionary.txt",
      "description": "Short description of the
        dictionary, this will be
        shown in the UI"
    }
  ]
}

```

Property	Type	Description	Required
id	String	Identifier of the module, must be the same as the python filename of the module.	Yes
className	String	Name of the class for the Plugin module.	Yes
name	String	Display name of the module inside the tool.	Yes
developer	String	Display name of the target software that the module tries to decrypt.	No
category	String	Display name of the category that the target software belongs to.	Yes
enabled	Boolean	Flag to enable\disable the module.	Yes
platform	Enumerator: [WINDOWS, ANDROID, MEMORY]	Platform of the target system that the module will extract data from.	Yes
package	String	Android package of the module under analysis (Android only).	No
view_extension	String or NULL	Identifier linking to the module's view extension (Android only).	No

Table 4.2: Module fields in AWLPS configuration file.

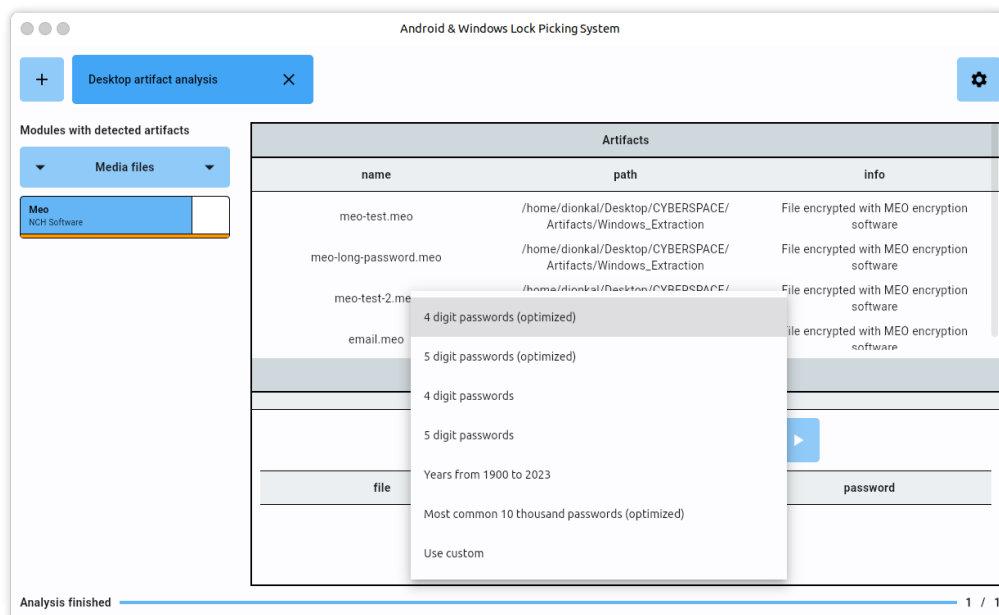


Figure 4.2: Dialogue selection for dictionaries inside the AWLPS tool.

Finally, as the last part of the configuration file, dictionaries contain a list of all the available dictionaries on the system that the tool can utilize to brute force passwords and hashes. For each dictionary registered, the configuration file has information about the path and the name of the dictionary file along with a short description of the dictionary to display inside the tool as seen in Figure 4.2.

4.2.2 User Interface

We have designed AWLPS from the beginning to be an application with a **Graphical User Interface (GUI)**. Nowadays, it is very important for a security tool of this scope to offer an “*easy-to-use*” GUI as all the aforementioned tools in section 4.1 have.

Digital forensics investigators often have limited time to spend on a case due to the large number of cases. Therefore the tools that will help them the most are tools that are easy and quick to use that have an easy learning curve to use and will boost the productivity and efficiency of the user.

Below we present a use case of AWLPS by analyzing a windows filesystem while simultaneously showcasing the GUI. Figure 4.5, displays the menu tree of AWLPS with all the different operations and actions that a user can do.

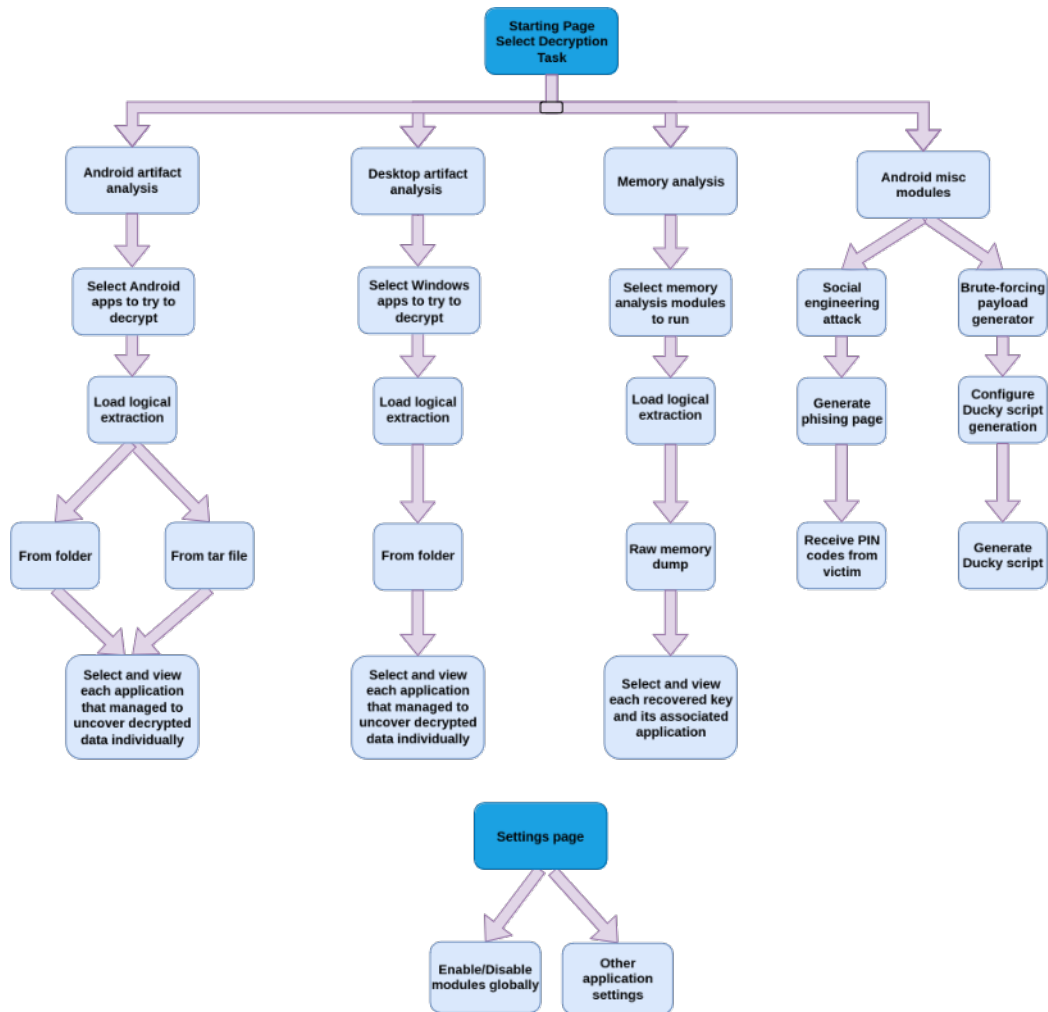


Figure 4.3: Menu Tree of the AWLPS tool.

4.2.2.1 AWLPS Analysis of Windows Artifacts

At startup the first thing the investigator that uses AWLPS would see is the landing page depicted in Figure 4.4. The user is able to choose between 4 different capabilities of the tool:

1. Windows encrypted artifact analysis
2. Windows Memory analysis
3. Android encrypted artifact analysis
4. Miscellaneous Android modules

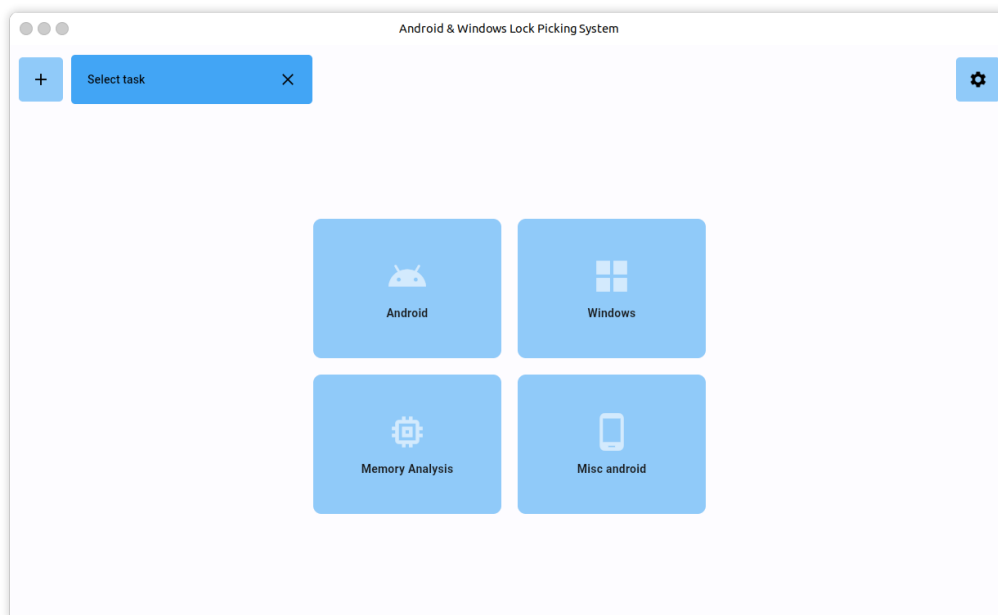


Figure 4.4: AWLPS landing page.

Next, by clicking the “Windows” button, we are redirected to the page shown in Figure 4.6. This page is responsible for all the encryption workarounds that perform **Filesystem Analysis** in Windows as explained in Chapter 3.1. The scope this thesis excludes the Android platform which was previously implemented and documented by Basha [3].

On the top left side of the screen, there is the section to load the input files to perform the encryption workarounds. For the time being, there are two ways of loading files, either by providing a path to a logical extraction¹⁹ by clicking the “From folder” button or by providing an “.E01” file (EWF) that the tool automatically mounts as a read only filesystem by clicking the “Mount EWF file” button. In both cases, a new window dialog launches as seen in Figure 4.5 and prompts the user to select the path of the folder or EWF file using the native file browser of the operating system.

Below the “Source Selection” buttons, there is the “Modules” section, which contains all the available encryption workaround modules that the tool is able to perform for the Windows platform. The modules are placed based on the category which they belong to, in our example there are two categories:

1. Media files: Meo encryption software
2. Browsers: Firefox, Chrome, Opera, Edge

¹⁹Read more about logical extractions in Fukami et al. [25]

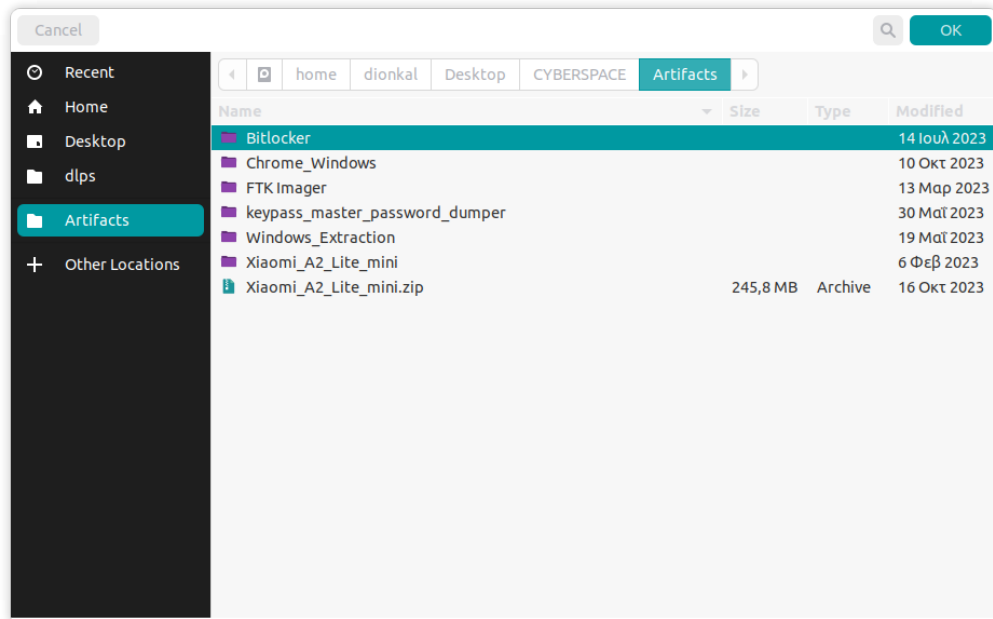


Figure 4.5: User input dialog in AWLPS to load an EWF file.

For each module we can see the name of the software it decrypts data and optionally its developer as they are defined in the configuration file. Additionally, we can enable or disable each module by selecting the checkbox on the left of the module. Doing this will determine if the module will run or not when we load an input source.

Finally, after choosing which modules to run and loading an input source by clicking either button, AWLPS will automatically run each enabled module for the loaded input. After the analysis finishes, we will be able to examine the results of each module separately as seen in Figure 4.8. In the side panel on the left we can see modules that AWLPS run for the current analysis, again separated by category. Each module will have a colour based on the status of the decryption:

- Green: The module successfully managed to decrypt data of the target software.
- Orange: The module managed to locate artifacts of the targeted software but manual user intervention is required to decrypt them. For example in the Meo module it requires the user to select a wordlist to perform a dictionary attack as described in Chapter 3.1.3.3.
- Cyan: Target software data were not found in the given input. This usually means that the software application is not installed in the system that we performed analysis.

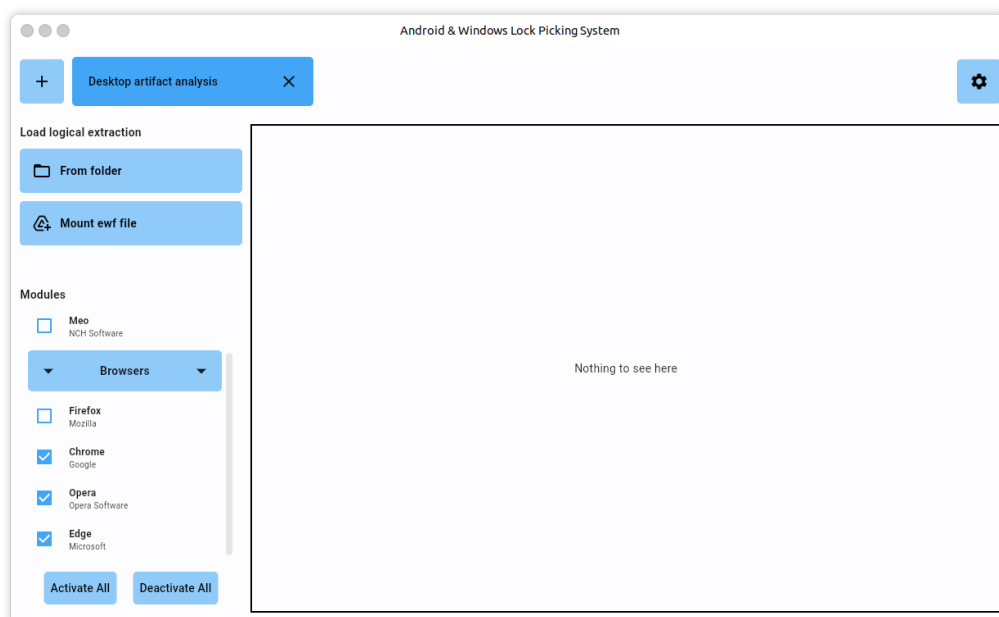


Figure 4.6: AWLPS analysis of Windows encrypted artifacts.

- Red: The module encountered an error while trying to decrypt data from the target software, as seen in Figure 4.7. In that case the module developer should look at the AWLPS log files²⁰ of the application to see more details about the error of the module.

Modules that are Orange or Green can be interacted with. By clicking on them we can see in the center of the screen more details for the analysis of each module.

In Figure 4.8 we have clicked on the Firefox module on the left and therefore we see details about the module. More specifically in the top panel we can see all the decrypted credentials that were stored by Mozilla software on our system and on the bottom panel we see details about the image integrity of our input. We can verify that the input did not change by comparing the before and after MD5 and SHA256 hashes. This type of output functionality is available on all implemented modules in our tool.

4.2.2.2 AWLPS Memory Analysis

Similar to the encrypted analysis of Windows artifacts, to perform an analysis of memory is similar.

The tool opens in the landing page again as seen in Figure 4.4. Then, by clicking the “Memory Analysis” button we navigate to the memory analysis screen as seen in Figure 4.9 where we select which modules to run (i.e. BitLocker, KeePass).

²⁰Application logs are found in the folder `<project_name>\out\awlps_<timestamp>.log`

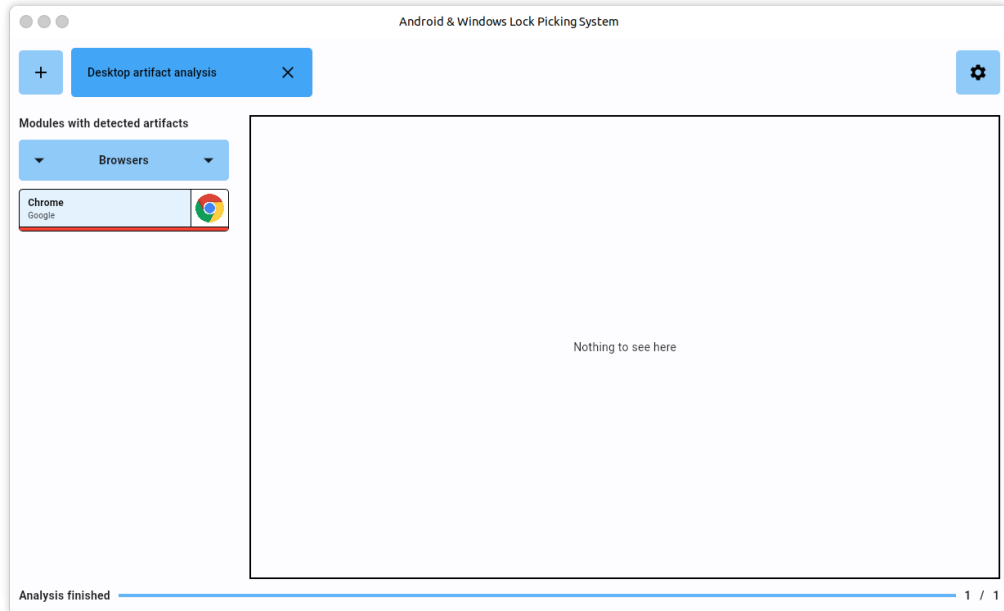


Figure 4.7: AWLPS instance of a failed module displayed in red.

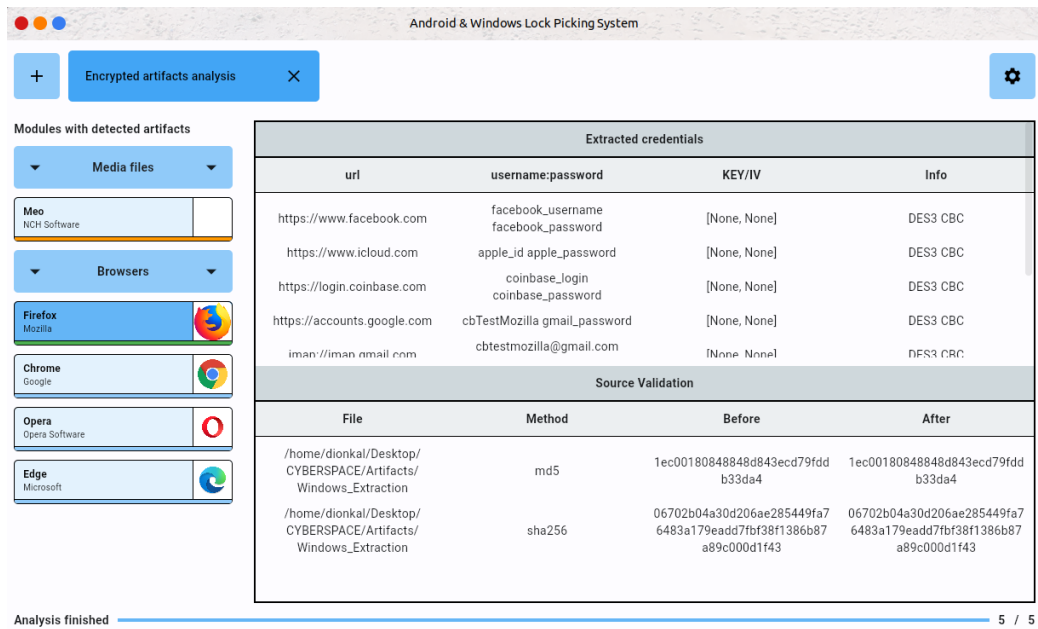


Figure 4.8: Details of the mozilla decryption module in AWLPS.

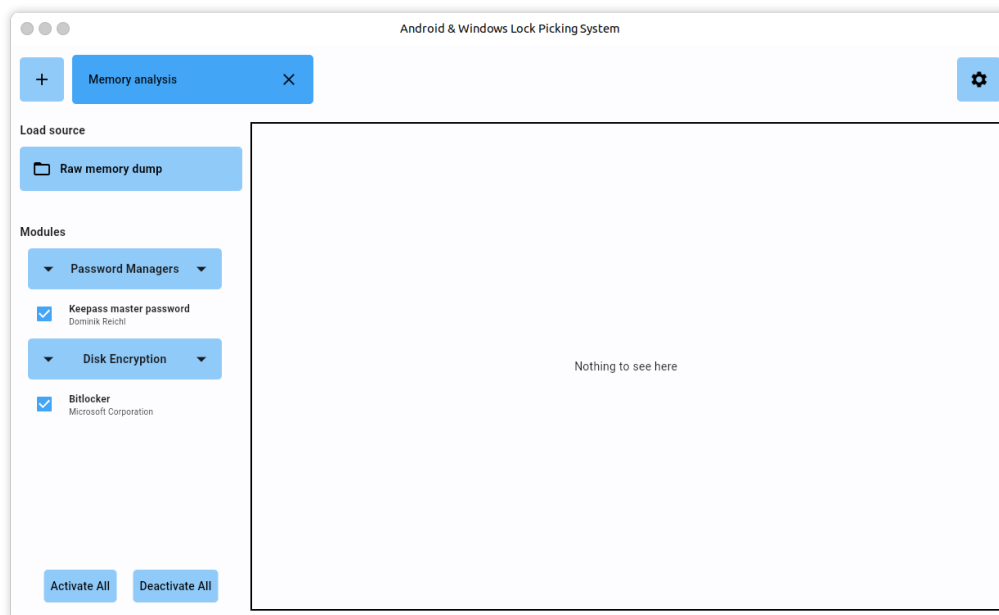


Figure 4.9: AWLPS memory analysis screen.

Currently, in memory analysis there is only one method of user input; loading memory files since volatility has the ability to process different file types internally through memory layers [70].

After selecting which memory modules to run and loading a memory file, AWLPS will run all the selected modules and try to decrypt the located encrypted artifacts. After the analysis has completed, it will display on the left the results for each module run, as observed in Figure 4.11. In a similar manner to the Windows artifact analysis the colour of each module depends on whether the module was successful, more specifically:

- Green: The module successfully managed to decrypt data of the target software.
- Orange: The module managed to locate artifacts of the targeted software but manual user intervention is required to decrypt them. For example in the Meo module it requires the user to select a wordlist to perform a dictionary attack as described in Chapter 3.1.3.3.
- Cyan: Target software data were not found in the given input. Target software data were not found in the given input. This usually means that the software application is not installed in the system that we performed analysis.
- Red: The module encountered an error while trying to decrypt data from the target software, as seen in Figure 4.7. In that case the user should look at

the logs of the application to see more details about the error of the module.

4.2.3 Volatility 3 Integration

Volatility 3, a renowned open-source tool, plays a crucial role in digital forensics investigations, particularly in memory analysis. The formatting of memory varies between files and can even change across different versions of the same operating system. To address these challenges, Volatility 3 incorporates **Memory Layers**, allowing it to seamlessly handle input from diverse data sources without burdening the user with the conversion overhead.

Recognizing its utility, we opted to integrate Volatility 3 into our toolkit, leveraging its option to be used as a library by other tools. To streamline the process, we developed **Volatility Wrapper** that plugins can utilize to import, set up and run volatility with ease.

Additionally, we have created **Volatility3-Bitlocker**²¹, a plugin specifically designed for Volatility 3. This plugin extracts the BitLocker Full Volume Encryption Key (FVEK) from memory, with the memory pool tag scanning technique, further elaborated in *Chapter 3*. Figure 4.10 depicts the output of the plugin that scanned a Windows 10 memory dump that was using AES-XTS 128 bit mode to encrypt the volume with FVEK being the hex value located at the second column of the output.

```

dionkal develop 3 ~ repos > volatility3 SIGINT python3 vol.py -f ~/Desktop/CYBERSPACE/Arti
facts/Bitlocker/bitlocker_memdump.mem bitlocker
Volatility 3 Framework 2.4.2
Progress: 100.00          PDB scanning finished
Cipher  FVEK      TWEAK Key
AES-XTS 128 bit (Win 10+)  a482fc60c844c79c6c6e5143e8076e8f      Not Applicable

```

Figure 4.10: BitLocker FVEK extraction from acquired memory using Volatility 3.

Figure 4.11 showcases the integration of Volatility 3 and our custom plugin to our tool AWLPS. On the center of the screen the panel “Possible BitLocker Full Volume Encryption Key” contains information about the possible keys it found. For example, the path of the analyzed memory file that we found the key, the encryption mode that the key is used to decrypt the drive and finally the key itself.

The panel “Source Validation” right below, contains all the information necessary to ensure the **forensic integrity** of the BitLocker decryption module. More specifically, it contains the path and the name of the analyzed memory file, the validation method (MD5 and SHA256 in this case) and finally, the values from before and after the analysis in order to validate that the image did not change.

²¹See <https://github.com/Dionkal/volatility3-bitlocker>

The screenshot shows the AWLPS interface with the following components:

- Header:** Android & Windows Lock Picking System
- Navigation:** + Memory Analysis X (Close) [Settings]
- Modules with detected artifacts:**
 - ▼ Disk Encryption ▼
 - BitLocker (Microsoft Corporation)
- Possible BitLocker Full Volume Encryption Key Table:**

Source	Mode	Value
/home/dionkal/Desktop/CYBERSPACE/Artifacts/Bitlocker/bitlocker_memdump.mem	AES-XTS 128 bit (Win 10+)	a482fc60c844c79c6c6e5143e8076e8f
- Source Validation Table:**

File	Method	Before	After
/home/dionkal/Desktop/CYBERSPACE/Artifacts/Bitlocker/bitlocker_memdump.mem	md5	d0dd72cdcc434e1b091c671f54cbc776	d0dd72cdcc434e1b091c671f54cbc776
/home/dionkal/Desktop/CYBERSPACE/Artifacts/Bitlocker/bitlocker_memdump.mem	sha256	ba0dd39a711b79b15cf8edb0d124ae8e0c61d7b768ad157b95bd5741fca64d9d	ba0dd39a711b79b15cf8edb0d124ae8e0c61d7b768ad157b95bd5741fca64d9d
- Status:** Analysis finished 1 / 1

Figure 4.11: Extraction of BitLocker FVEK from the AWLPS.

Chapter 5

Conclusion

In this chapter we discuss the limitation of our tool and present future extensions that will address them. Additionally, we include an “Ethics” section to provide a disclaimer for our work. Finally, we conclude by summarizing the key findings and contributions of this thesis.

5.1 Future work

Our current efforts have centered around establishing the foundational elements of a security tool capable of decrypting data across various applications. With the core functionality now completed, our attention turns to enhancing the tool’s usability by incorporating additional features. Below, we outline the key features that we are working to integrate into our tool, AWLPS.

5.1.1 Forensic Image Loading & Handling

Currently, AWLPS doesn’t do any processing of forensic images itself. Rather the investigator should have previously mounted the forensic image to their workstation, in order to perform analysis and decrypt the artifacts. Our aim, is to implement an image utility handler inside the tool that will offload the task of mounting, handling and dismounting a forensic raw (.dd) and expert witness format (efw, E01) image from the investigator.

5.1.2 Additional Workarounds

We designed AWLPS to be extendable as mentioned in Chapter 4. Subsequently we would like to add more encryption workaround modules for all the supported platforms. Currently, the tool supports 20 modules for Android, 5 modules for the Windows file system and 2 for Windows memory.

Additionally, we want to expand our existing research mentioned in Chapter 3. More specifically, we would like to support the decryption of artifacts of the

methodologies we studied for other operating systems as well such as Linux and MacOS.

Some encryption workarounds such as the “**Mozilla Decryption**” (see Chapter 3.1.1), function as intended in Linux and MacOS systems. Other workarounds that rely on OS libraries might have an entire different process for storing and accessing encryption keys. For example, the encryption of the web logins in the Chrome browser has an entire different process that relies in keychains as opposed to the Windows Data Protection API. To perform the chrome encryption workaround in Linux we would need to investigate the Linux PAM subsystem as discussed in Chapter 3.1.2.1 to subvert it.

5.1.3 Reporting & Documentation

Being able to export data in a report from the tool is very important in forensic software, as we examined in Chapter 4. Our goal is to improve the reporting capability of AWLPS by exporting data such as application artifacts, decrypted file data and metadata such as forensic validations in a file. Then, these data can be imported easily into other forensic security software for further forensic analysis such as Autopsy. In order to streamline this process, we would need to create and export files from our tool in JSON and csv. Moreover, we plan to support the option to export to PDF so that the findings from our tool can be integrated in the investigator’s forensic report.

Additionally, we aim to incorporate in-app user documentation for each encryption workaround module. This feature is designed to facilitate investigators in understanding the purpose and functionality of each module, providing a clear understanding of their operations.

5.1.4 User Testing

Generally, user testing is an integral part of the software development process. It allows the developers to refine the software, harden it by squashing bugs and making quality-of-life changes by getting feedback from the user’s usage. In our case, we hope to distribute a demo version of our tool in various LEAs that are members of our project consortium to gain valuable feedback directly from the digital forensics investigators that have to deal with the encryption problem all the time.

The feedback we will be aiming to get is on how to improve the **user experience** of AWLPS by fixing any bugs that the users might encounter. We also hope to get suggestions from the investigators and from new case study reports like Europol’s IOCTA¹, on what new encryption workaround modules we should focus our research on.

¹Europol releases each year an IOCTA report, our research was based on the reports of 2014-2021 but newer reports might uncover new criminal trends, <https://www.europol.europa.eu/publications-events/main-reports/iocta-report>

5.2 Ethics

It is important to stress out that the encryption workarounds we presented in our work were already publicly known, thus our contributions in this thesis would not further deteriorate the security of the discussed software.

Subsequently, we tailored AWLPS to exclusively function with local forensic copies. To decrypt the data, investigators must have previously seized the suspect's physical device, usually by obtaining a warrant first. This limits the intrusion to the suspect's device only and ensures the privacy and security of lawful citizens remain unchallenged. This approach effectively addresses concerns about citizens' right to privacy by adhering to the principle of **proportionality** described in Chapter 2.2.3.

5.3 Summary

Encryption offers protection through authentication and transport secure protocols and safeguards citizens privacy by encrypting their communications from third parties. Similarly, criminals benefit of encryption as well by protecting their communications from law enforcement.

In this thesis, we discussed the proposed solution of encryption workarounds introduced by Kerr and Schneier [39]. Additionally, we presented encryption workarounds for desktop operating systems and we validated that are forensically sound. Finally, we transformed ALPS into AWLPS, a modular tool designed to execute encryption workarounds, thereby enabling law enforcement to enhance their decryption capabilities.

Appendix A

Appendix A

A.1 Taxonomy of encryption workarounds included in AWLPS

The taxonomy of encryption workarounds presented by Kerr & Schneier in [39] currently supported in AWLPS. The addition of this thesis in the taxonomy is the following:

- **Find the key:**

1. Mozilla Firefox, Thunderbird and TOR
2. BitLocker
3. Google Chrome
4. Opera Browser
5. Microsoft Edge

- **Guess the key:**

1. Meo encryption software

- **Exploit a flaw:**

1. KeePass 2, CVE-2023-32784

The rest of the encryption workarounds related to Android are discussed in [3].

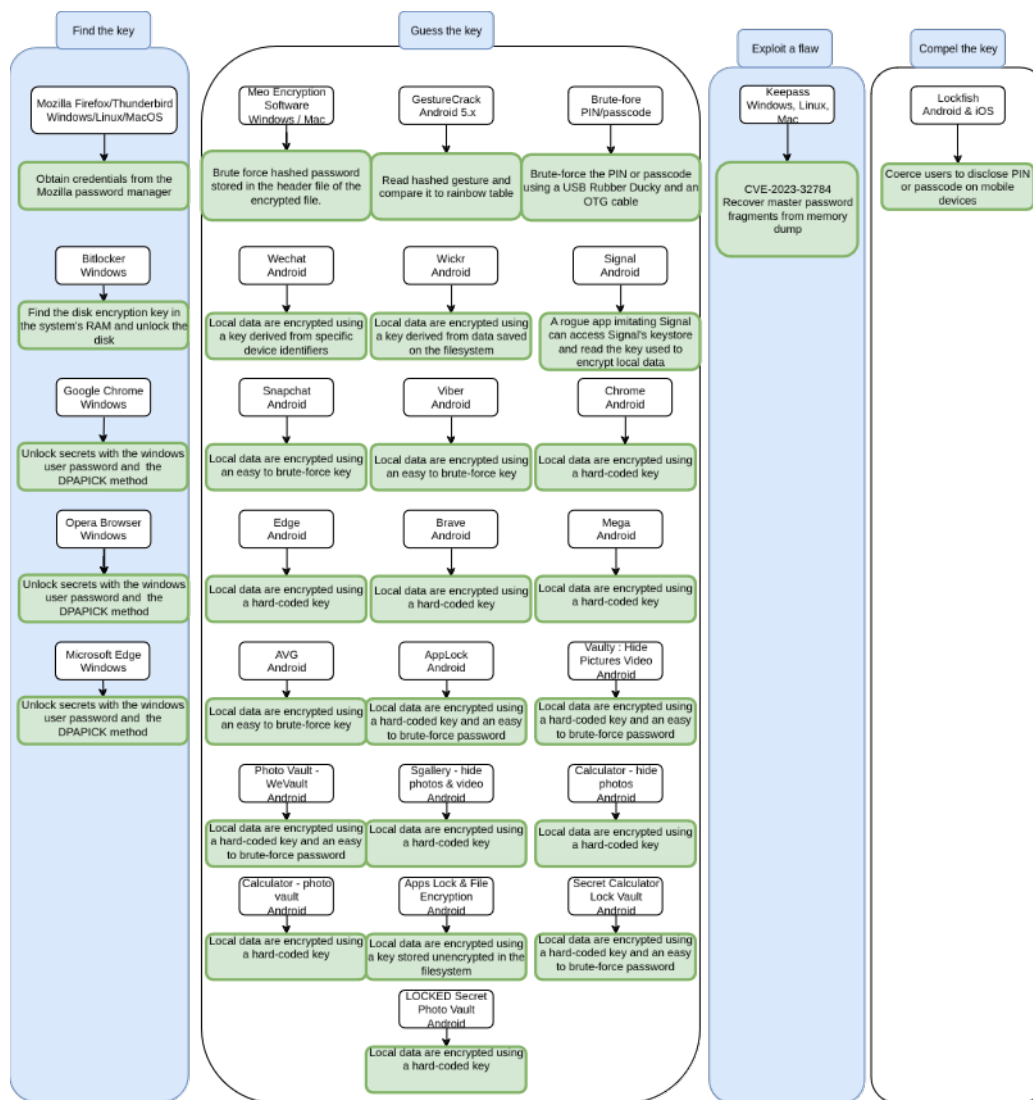


Figure A.1: List of all the modules offered in AWLPS based on each category introduced by Kerr & Schneier [39].

A.2 Forensic Drive Acquisition using Guymager

To perform the experiment of unlocking a BitLocker encrypted drive we needed to perform a physical acquisition of the encrypted drive. Although in a real digital forensics case the investigators would use more specialized tools to perform a device acquisition (i.e. hardware write blockers ¹).

We on the other hand, lacking the necessary resources we used **Guymager** ² to do this, a software acquisition tool that comes installed with Kali Linux. Below are screenshots from the Guymager tool during the acquisition of a BitLocker encrypted drive.

Figure A.2 depicts the drive selection before performing the acquisition. Guymager offers two acquisition formats:

1. Linux dd raw image which performs an exact copy bit to bit to the destination device
2. Expert Witness Format (EWF) which compresses the image and has important metadata for forensic investigations, such as:
 - Case number
 - Evidence number
 - Examiner that performed the acquisition
 - Description of the evidence (i.e. where it was found)
 - Note, additional information that might be useful to the investigator (i.e the machine was running at the time of the acquisition)
 - Hash values of the acquired device to perform data verification

¹See NIST specifications of a Hardware Write Blocker device, <https://www.nist.gov/system/files/documents/2017/05/09/hwb-v2-post-19-may-04.pdf>

²See <https://www.kali.org/tools/guymager/#tool-documentation>

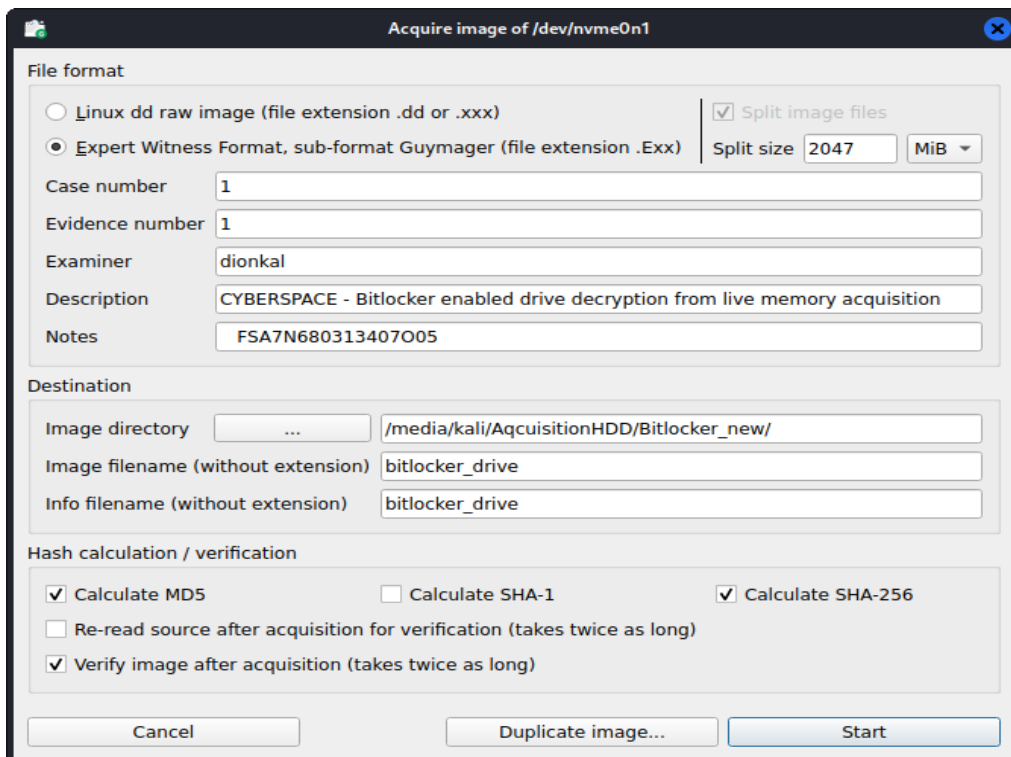


Figure A.2: Guymager - drive selection

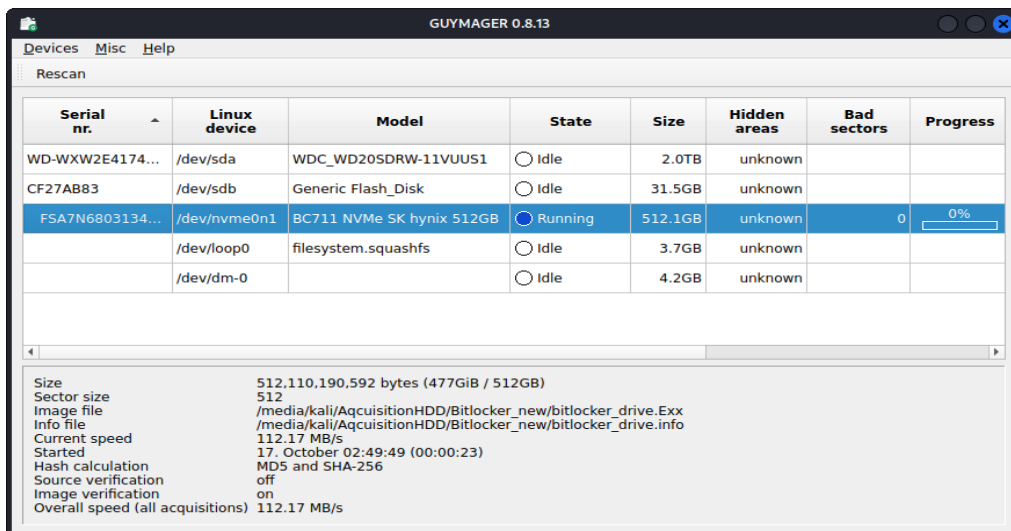


Figure A.3: Guymager - acquisition process

Appendix B

AWLPS Installation and Extension

B.1 AWLPS Installation & Configuration

B.1.1 Linux

To setup AWLPS in Linux you will need the following, note that we used Ubuntu Linux 22.04, if you have a different distribution some commands might change.

Requirements:

- git (comes preinstalled in most Linux distributions)
- Python 3 (3.10 or higher)
`sudo apt install python3`
- Python virtual environment (optional)
`python3 -m pip install virtual-env`
- libsqlcipher
`sudo apt install libsqlcipher-dev`
- libewf
`sudo apt install ewf-tools`

Setup:

1. Download AWLPS from the Github repository ¹.
`git clone git@github.com:Dionkal/dlps.git`

¹Github repository, note that it requires authorization from owner, <https://github.com/Dionkal/dlps>

2. (Optional) Setup a virtual environment:

(a) Create a virtual environment:

```
python3 -m venv <virtual_environment_name>
```

(b) Activate the virtual environment:

```
source <virtual_environment_name>/bin/activate
```

3. Install python dependencies:

- `python3 -m pip install -r requirements.txt`
- (Optional) Install optional dependencies. Some modules (e.g. modules that require volatility) might not work if they are not installed.
`python3 -m pip install -r requirements_optional.txt`

4. Start the tool:

```
$ ./awlps.sh
```

B.1.2 Windows

To setup AWLPS in Windows you will need the following:

Requirements:

- git: <https://git-scm.com/download/win>
- Python 3 (3.10 or higher): <https://www.python.org/downloads/windows/>
- Python virtual environment (optional)
`python3 -m pip install virtual-env`
- libsqlcipher
- libewf:
Download from here <https://sourceforge.net/projects/libewf/>

Setup:

1. Download AWLPS from the Github repository ².
`git clone git@github.com:Dionkal/dlps.git`

2. (Optional) Setup a virtual environment:

(a) Create a virtual environment:

```
python -m venv <virtual environment name>
```

²Github repository, note that it requires authorization from owner, <https://github.com/Dionkal/dlps>

(b) Activate the virtual environment:

```
<virtual environment name>\Scripts\Activate.bat
```

3. Install python dependencies:

- `python3 -m pip install -r requirements.txt`
- (Optional) Install optional dependencies. Some modules (e.g. modules that require volatility) might not work if they are not installed.
`python3 -m pip install -r requirements_optional.txt`

4. Start the tool:

```
$ ./awlps.sh
```

B.2 AWLPS Plugin Extension

In this section we document the extensibility of AWLPS. We have created a simple scenario of an encryption program that we want to perform encryption workarounds. Below we demonstrate how to integrate the workaround into AWLPS.

B.2.1 Scenario

Assume **Demo** is a simple encryption software that encrypts files locally. A forensic examiner has found that Demo is used in a case they are investigating. They have also found a weakness in the encryption process and managed to decrypt data encrypted with Demo. To integrate the workaround into AWLPS the following steps must be completed.

B.2.2 Implementation

Create a class that extends the base `Plugin` class. In this example we will name it `DemoDecryption` .

```
class DemoDecryption (Plugin):
```

In the `DemoDecryption` there are 3 items need to be defined:

- **Decryption**

The `_decrypt` method is the core function of the plugin. It is responsible for performing the **encryption workaround** and decrypting the target application artifacts.

To do this the method takes as input the `source` which is a string containing the path of the data source. After which, the encryption workaround technique should be performed to locate and decrypt the required artifacts.

Finally, `_decrypt` should return a `Tuple` that contains:

1. a list of all the detected **Artifacts** such as files, encryption algorithms and anything else used in the decryption process of the application
2. a list with all the decrypted data as strings

```
def _decrypt(self, source: str)
-> Tuple[List[Artifact], List[str]]:
    # Write decryption code here
```

The next code block shows how to create a new `Artifact` object. It requires 3 arguments:

1. **name**: the name of the artifact (i.e. filename)
2. **path**: the relative path from the source that the artifact is located
3. **info**: anything of importance related to the artifact, (i.e how the file is used in the decryption)

```
new Artifact(name: str, path: str, info: str)
```

• Validation

Apart from the decryption, each module is required to define the validation methods of the input source to guarantee forensic soundness. To do this, the `_validators` object inside the `DemoDecryption` should be initialized as follows:

```
_validators = {  
    "md5": lambda source: return <md5_hash_in_string>,  
  
    "sha256": lambda source: return <sha256_hash_in_string>  
}
```

The `_validators` object can have any string as key. It should be named accordingly to display the validation algorithm or method used. The value is always a function that takes as argument the path of the data **source** used in the current decryption plugin. Note that each validation function defined inside the `_validators` object will run twice during the execution of the `DemoDecryption` plugin, once before the decryption starts and then after the decryption finishes. The plugin will automatically validate the source by comparing the validation values before and after the decryption, if there is any mismatch an error will be raised in the log files.

• Render

The `render` method is responsible for handling the decryption results of the plugin and organizing the output for display back in the GUI.

```
def render(self, output: Output) -> ModuleStatus:
```

The `output` argument is the GUI component that manages how each module will display its results.

The `render` function returns a `ModuleStatus` object which is an enumerator for the status of the module, make sure to return the appropriate `ModuleStatus` as shown below.

```
class ModuleStatus(Enum):
    """
        Enumerator for the Plugin output status after it run.
    """
    FAILED = 0      # The plugin Encountered an error
    NOT_FOUND = 1   # The plugin run but didn't find anything
    SUCCEEDED = 2   # The plugin run and found results, should have decrypted data
    MANUAL = 3      # The plugin run, found important data, user attention required
```

Finally, the plugin code should look like the following:

```
import logging

from typing import List, Tuple
from utils.artifact import Artifact
from utils.plugin import Plugin, ModuleStatus
from panels.artifact_analysis.output import Output

logger = logging.getLogger('alps')

class DemoDecrypt(Plugin):

    _validators = {
        "md5": <md5 file validator>,
        "sha256": <sha256 file validator>
    }

    def _decrypt(self, source) -> Tuple[List[Artifact], List[str]]:
        artifacts = []
        results = []

        # Perform encryption workaround
        # add related files and cryptographic
        # primitives to the artifacts list
        # decrypted data should be added to
        # the results list

        return artifacts, results

    def render(self, output: Output) -> ModuleStatus:
        status = ModuleStatus.NOT_FOUND
        try:
            for source in self._sources.items():
                if source[1]['artifacts']:
```

```
status = ModuleStatus.SUCCEEDED
output.add_section('Extracted credentials',
                  ['url', 'username:password', 'KEY/IV', 'Info'])

for data in source[1]['data']:
    output.append_text_to_section('Extracted credentials',
                                data)

output.add_section('Source Validation', ['File',
                                       'Method', 'Before', 'After'])

for name, validation in source[1]['validate_obj']
    .validation.items():
    output.append_text_to_section(
        'Source Validation', [
            source[0], name, validation['before'],
            validation['after']])

except Exception as e:
    logger.error(f'Error: {e}')
    return ModuleStatus.FAILED

return status
```


Bibliography

- [1] Aorimn. dislocker. github.com/Aorimn/dislocker.
- [2] *Introduction to ASN.1 Syntax and Encoding*.
- [3] Skerdi Basha. Encryption workarounds for android, 2023.
- [4] Corentin Bayet and Paul Fariello. Scoop the windows 10 pool! In *Proceedings of SSTIC*, 2020.
- [5] Natasha Bertrand. The fbi staged a lovers' fight to catch the kingpin of the web's biggest illegal drug marketplace. *Business Insider available at www.businessinsider.com/ross-ulbricht-will-be-sentenced-soon-heres-how-he-was-arrested-2015-5*, 2015.
- [6] breppo. Volatility_bitlocker. github.com/breppo/Volatility-BitLocker/tree/master, 2020. Last commit hash: 4dc7e17df0557c4e18ef7fca685f8a1ad416128f.
- [7] Dominic Casciani and Gaetan Portal. Phone encryption: Police 'mug' suspect to get data. *BBC available at www.bbc.com/news/uk-38183819*, 2016.
- [8] The Digital Forensic Research Conference. A road map for digital forensic research. In *Report From the First Digital Forensic Research Workshop*, 2001.
- [9] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and Xiaofeng Wang. The tangled web of password reuse. In *The Tangled Web of Password Reuse*, 01 2014.
- [10] Geplaatst door. Dpapi-in-depth with tooling: standalone dpapi. www.insecurity.be/blog/2020/12/24/dpapi-in-depth-with-tooling-standalone-dpapi/.
- [11] EC3. Internet organised crime threat assessment 2015. Technical report, Europol, 2015.
- [12] EC3. Internet organised crime threat assessment 2016. Technical report, Europol, 2016.

- [13] EC3. Internet organised crime threat assessment 2017. Technical report, Europol, 2017.
- [14] EC3. Internet organised crime threat assessment 2018. Technical report, Europol, 2018.
- [15] EC3. Internet organised crime threat assessment 2019. Technical report, Europol, 2019.
- [16] EC3. Internet organised crime threat assessment 2020. Technical report, Europol, 2020.
- [17] EC3. Third report of the observatory function on encryption. Technical report, Europol, EUrojust, 2021.
- [18] Reed Albergotti Ellen Nakashima. The fbi wanted to unlock the san berardino shooter's iphone. it turned to a little-known australian firm. *The Washington Post*, 2021.
- [19] 2016. Encryption of data - Questionnaire, Council of the European Union, Original questionnaire can be found at data.consilium.europa.eu/doc/document/ST-12368-2016-INIT/en/pdf, country responses can be found at www.asktheeu.org/en/request/input_provided_by_ms_on_question?nocache=incoming-11727#incoming-11727.
- [20] 'Electronic evidence is needed in around 85% of criminal investigations, and in two thirds of these investigations there is a need to obtain evidence from online service providers based in another jurisdiction'. Recommendation for a Council Decision authorising the opening of negotiations in view of an agreement between the European Union and the United States of America on cross-border access to electronic evidence for judicial cooperation in criminal matters, COM(2019) 70 final, 1.
- [21] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 657–666, New York, NY, USA, 2007. Association for Computing Machinery.
- [22] Foxtan Forensics. Analysing chrome login data. www.foxtonforensics.com/blog/post/analysing-chrome-login-data, 2019. Accessed: 2023-03-8.
- [23] Lorenzo Franceschi-Bicchierai. Facebook helped the fbi hack a child predator. *Vice*, 2020.
- [24] Yair Frankel and Moti Yung. Escrow encryption systems visited: Attacks, analysis and designs. In Don Coppersmith, editor, *Advances in Cryptology — CRYPTO' 95*, pages 222–235, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

- [25] Aya Fukami, Radina Stoykova, and Zeno Geradts. A new model for forensic data extraction from encrypted mobile devices. *Forensic Science International: Digital Investigation*, 38:301169, 2021.
- [26] g0tmi1k. Kali linux forensics mode. Technical report, Offensive Security, 2023.
- [27] GCHQ. The equities process. www.gchq.gov.uk/information/equities-process. Accessed: 2023-1-10.
- [28] Petter Gottschalk. *Knowledge management systems in law enforcement : technologies and techniques*. Hershey, PA : Idea Group Pub., 2007. ark:/13960/s21vc99kjxb.
- [29] www.kali.org/tools/guymager/.
- [30] Hashcat. github.com/hashcat/hashcat.
- [31] Hakan Hekim, Serdar Kenan Gul, and Bahadir K. Akcam. Police use of information technologies in criminal investigations. 2013.
- [32] White House. Vulnerabilities equities policy and process for the united states government. trumpwhitehouse.archives.gov/sites/whitehouse.gov/files/images/External%20-%20Unclassified%20VEP%20Charter%20FINAL.PDF. Accessed: 2023-10-10.
- [33] Information Commissioner's Office, Principle (c): Data minimisation ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/data-protection-principles/a-guide-to-the-data-protection-principles/the-principles/data-minimisation/.
- [34] Interpol. Global operation targets child sexual abuse material exchanged via messaging apps. www.interpol.int/en/News-and-Events/News/2017/Global-operation-targets-child-sexual-abuse-material-exchanged-via-messaging-apps 2017.
- [35] INTERPOL. Global guidelines for digital forensics laboratories. Technical report, INTERPOL, 2019.
- [36] RFC: Internet Relay Chat Protocol.
- [37] Internal Security Fund Performance: commission.europa.eu/strategy-and-policy/eu-budget/performance-and-reporting/programme-performance-statements/internal-security-fund-performance_en.
- [38] Keepass 2.45 released. keepass.info/news/n230603_2.54.html, 2023. Accessed: 2023-10-10.

- [39] Orin S Kerr and Bruce Schneier. Encryption workarounds. *Geo. LJ*, 106:989, 2017.
- [40] libyal. libewf. github.com/libyal/libewf.
- [41] Michael Hale Ligh, Andrew Case, Jamie Levy, and Aaron Walters. *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. Wiley Publishing, 1st edition, 2014.
- [42] Rodney McKemmish. When is digital evidence forensically sound? In Indrajit Ray and Sujeet Shenoi, editors, *Advances in Digital Forensics IV*, pages 3–15, Boston, MA, 2008. Springer US.
- [43] Microsoft. Bitlocker settings reference. learn.microsoft.com/en-us/mem/configmgr/protect/tech-ref/bitlocker/settings.
- [44] Microsoft. CryptProtectData function (dpapi.h). learn.microsoft.com/en-us/windows/win32/api/dpapi/nf-dpapi-cryptprotectdata.
- [45] Microsoft. Getkeyprotectortype method of the win32.encryptablevolume class. learn.microsoft.com/en-us/windows/win32/secprov/getkeyprotectortype-win32-encryptablevolume.
- [46] Microsoft. Overview of bitlocker device encryption. learn.microsoft.com/en-us/windows/security/operating-system-security/data-protection/bitlocker/bitlocker-device-encryption-overview-windows-10.
- [47] Dan Milmo. Apple suggests imessage and facetime could be withdrawn in uk over law change. *The Guardian* www.theguardian.com/technology/2023/jul/20/uk-surveillance-law-changes-could-force-apple-to-withdraw-security-features, 2023.
- [48] Robert Morris and Ken Thompson. Password security: A case history. *Commun. ACM*, 22(11):594–597, nov 1979.
- [49] *Metasploit Framework: Post Modules, Documentation available at docs.metasploit.com/docs/pentesting/metasploit-guide-post-gather-modules.html*.
- [50] Rebecca Nelson, Atul Shukla, and Cory Smith. *Web Browser Forensics in Google Chrome, Mozilla Firefox, and the Tor Browser Bundle*, pages 219–241. Springer International Publishing, Cham, 2020.
- [51] National Institute of Standards and Technology. Nvd-cve-2023-32784. nvd.nist.gov/vuln/detail/CVE-2023-32784, 2023. Accessed: 2023-10-10.

- [52] Manhattan District Attorney's Office. Smartphone encryption and public safety. Technical report, 2019.
- [53] *RFC: PKCS #5: Password-Based Cryptography Specification, Chapter 5.2.*
- [54] Mike Power. A new robot dealer service makes buying drugs easier than ever. Technical report, Vice, 2020. www.vice.com/en/article/jgxyyp/televend-robot-drug-dealer-telegram.
- [55] RFC 3227: Guidelines for Evidence Collection and Archiving, 2.1 Order of Volatility, www.rfc-editor.org/rfc/rfc3227.
- [56] RFC 3826: The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model, <https://www.rfc-editor.org/rfc/rfc3826>.
- [57] RFC2631: Diffie-Hellman Key Agreement Method, www.rfc-editor.org/rfc/rfc2631.
- [58] RFC6979: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA), www.rfc-editor.org/rfc/rfc6979.
- [59] RFC 1321: The MD5 Message-Digest Algorithm, www.rfc-editor.org/rfc/rfc1321.
- [60] RFC3447: Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography www.rfc-editor.org/rfc/rfc3447.
- [61] RFC 3174: US Secure Hash Algorithm 1 (SHA1), www.rfc-editor.org/rfc/rfc3174.
- [62] RFC 7860: HMAC-SHA-2 Authentication Protocols in User-Based Security Model (USM) for SNMPv3, www.rfc-editor.org/rfc/rfc7860.
- [63] RFC 6234: US Secure Hash Algorithms, www.rfc-editor.org/rfc/rfc6234.
- [64] Rockyou dictionary. github.com/zacheller/rockyou.
- [65] Signal. Grand jury subpoena for signal user data, central district of california. signal.org/bigbrother/central-california-grand-jury/, 2021.
- [66] theincidentalchewtoy. Meo – file encryption software. theincidentalchewtoy.wordpress.com/2021/11/09/meo-file-encryption-software/, 2021. Personal blog of a digital forensics investigator. Accessed : 2023-03-23.
- [67] vdohney. Keepass discussion. sourceforge.net/p/keepass/discussion/329220/thread/f3438e6283/, 2023. Accessed: 2023-10-10.

- [68] vdohney. Keepass password dumper. github.com/vdohney/keepass-password-dumper, 2023. Last commit hash: dc13d44fae7fc94692bbe2a3e3862a5569d741f.
- [69] Verizon. 2023 data breach investigations report. Technical report, Verizon, 2023.
- [70] *Volatility 3 memory layers*, volatility3.readthedocs.io/en/latest/basics.html.
- [71] Chun Wang, Steve T.K. Jan, Hang Hu, Douglas Bossart, and Gang Wang. The next domino to fall: Empirical analysis of user passwords across online services. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY '18, page 196–203, New York, NY, USA, 2018. Association for Computing Machinery.
- [72] Kim Zetter. Security manual reveals the opsec advice isis gives recruits. *Wired* available at: www.wired.com/2015/11/isis-opsec-encryption-manuals-reveal-terrorist-group-security-protocols/, 2015.
- [73] André Årnes Stefan Axelsson Petter Christian Bjelland Ausra Dilijonaite Anders Orsten Flaglien Katrin Franke Jeff Hamm Jens-Petter Sandvik Inger Marie Sunde. *Digital Forensics*, chapter 1 Introduction, pages 1–11. John Wiley & Sons, Ltd, 2017.