Password Managers in Digital Forensics

Creating a Process to Extract Relevant Artefacts from Bitwarden and KeePass

Sascha Hähni

Department of Computer and Systems Sciences

Degree Project 30 HE Credits
Computer and Systems Sciences
Degree Project at the Master Level
Spring Term 2023
Supervisor: Stefan Axelsson

Stockholm University

Abstract

Digital forensics – the scientific process to draw evidence from digital devices confiscated in a criminal investigation – is constantly adapting to technological changes. A current challenge is the widespread use of encryption that makes classical data retrieval methods obsolete. Relevant data must now be retrieved from running devices and without delay, ideally directly at the time of seizure. This requires standardised processes and specialised tools to ensure no data is overlooked, that forensic integrity is maintained, and that encrypted data can be successfully made available to investigators. While research produced many promising results in this field in the last years, there is still much work to be done due to countless different applications, operating systems, and devices that all behave in different ways.

This thesis addresses a software category called password managers – applications that store login credentials to different services. Despite the obvious value of password manager data to a criminal investigation, a comprehensive description of a forensic process on how to extract such data has not yet been in the focus of research. The present work addresses this gap and presents a process to extract forensically relevant data from two password manager applications – *Bitwarden* and *KeePass* – by extending an existing forensic framework called *Vision*.

Using design science, a forensic extraction process was developed by thoroughly analysing the inner workings of the mentioned password managers. The artefact was named *Password Manager Forensics (PMF)* and consists of a four-step extraction process with different *Python* modules to automate the extraction of relevant data. *PMF* was tested against three scenarios in a laboratory setting to evaluate its applicability in an investigative context.

The results show that the artefact is able to extract forensically relevant information related to password managers that would otherwise not be readily available to investigators. *PMF* is capable to identify and extract relevant files, to extract master passwords from a memory dump, to parse configuration files for relevant data, to brute-force master passwords and PIN codes, to decrypt, extract, and validate password manager vault data, and to create summary reports.

PMF is the first comprehensive forensic process to extract relevant data from password managers. This brings new opportunities for digital forensics examiners and a potential to improve the handling of devices that contain password manager data in digital investigations.

The current version of *PMF* only supports *Windows* desktop applications of *Bitwarden* and *KeePass*. Yet, due to the open and flexible architecture of the artefact, further expansion and improvement is possible in future research.

Keywords: Digital Forensics, Password Managers, Memory Forensics, Live Forensics

Synopsis

Background: The growing use of encryption causes challenges to digital forensics. Researchers have therefore started to propose forensic processes that implement live forensics and memory forensics to extract data from encrypted seized devices. While there was a strong focus on device locks and disk encryption in the beginning, we now see more interest in memory and live forensics of specific applications.

Problem: As of today, there has not been much research on password managers although access to login credentials of a suspect could be of significant value in a forensic investigation.

Research Aim: Building upon existing work in the field of application-specific forensics, this thesis' aim is to develop a process on how to extract forensically relevant data from password managers.

Method: Following the design science framework, an artefact was developed. To do this, data about how password managers work was gathered from publicly available documents (security whitepapers), open-source code, and through observing the application in an experimental setting. Forensically relevant artefacts were then identified from this data. Building upon an existing framework called *Vision*, a process to extract the identified relevant artefacts from two password manager applications – *Bitwarden* and *KeePass* – was eventually developed. The artefact was tested in an artificial environment against scenarios modelled on real cases. Thereby, it was evaluated for its suitability to be used in investigations.

Result: The artefact was applied in three scenarios modelled on possible real investigations. In all three cases, it has been demonstrated that the artefact is able to extract forensically relevant information from password managers that would otherwise not be readily available to investigators. Among the artefacts capabilities are functions to identify and extract relevant files, to extract a master password as well as vault passwords from a memory dump, to parse configuration files for relevant data, to brute-force master passwords and PIN codes, to decrypt, extract, and validate password manager vault data, and to create summary reports.

Discussion: The presented artefact provides forensic examiners, for the first time, with a comprehensive tool to extract forensically relevant artefacts from password managers. The results have shown that, despite the high security standards of such applications, it is very often possible to extract artefacts relevant to a forensic investigation. Due to the limited resources of this thesis, the artefact currently only supports *Windows* desktop applications of two password managers. Yet, thanks to the open and modular architecture of the artefact, further expansion and improvement can be done in future research.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AES	Advanced Encryption Standard
API	Application Programming Interface
CBC	Cipher Block Chaining
CLI	Command Line Interface
FATKit	Forensic Analysis Toolkit
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HKDF	HMAC-based Key Derivation Function
HMAC	Hash-based Message Authentication Code
JSON	JavaScript Object Notation
MAC	Message Authentication Code
PBKDF2	Password-Based Key Derivation Function 2
PIN	Personal Identification Number
PMF	Password Manager Forensics
OS	Operating System
RAM	Random Access Memory
SaaS	Software-as-a-Service
TLS	Transport Layer Security
VM	Virtual Machine

1 Introduction

1.1 Background

Digital technology plays an increasingly important role in today's world. It is therefore not surprising that electronic devices are becoming more and more relevant in the investigation of crimes. The field that deals with such evidence is known as digital forensics. Digital forensics is concerned with the "use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived […] for the purpose of facilitating or furthering the reconstruction of events found to be criminal" [1, p. 4].

The forensic process involves identifying relevant data storage media related to a crime, as well as forensically securing (collecting) the data for further examination and analysis [2]. Traditionally, the focus of digital forensics was on artefacts located on digital storage devices like hard disks, memory cards, etc. that were seized and could later be analysed in a forensic laboratory. However, the increase in complexity of computer systems, the evolvement of volatile malware, and – most relevant for this thesis – the rise of data encryption resulted in a growing interest in live and memory forensics techniques [3].

Live and memory forensics allow to collect data that is not available in non-volatile storage, including "running processes, network connections, fragments of volatile data such as chat messages, and keying material for drive encryption" [3]. However, random access memory (RAM) data can only be obtained from a system that is turned on and running. Also, volatile memory is, as the name suggests, short-lived and fast to change. It is therefore important to acquire key information as soon as possible, ideally directly on the spot at search and seizure sites [4].

1.2 Problem

When collecting artefacts on the site of a crime, a forensic investigator needs to be very careful. This is to preserve the forensic integrity of all artefacts. But also, to prevent the loss of relevant (volatile) data. Investigators, therefore, often follow strict procedures to make sure all artefacts are handled properly, and relevant data is secured in accordance with protocol.

Intensive research is being conducted in the field of digital forensics in order to constantly improve data extraction and preservation processes [3]. In recent years, we have seen work in, among other areas, the extraction of hard drive encryption keys [5], [6], data extraction from

encrypted mobile devices [7], [8], or a comprehensive framework for artefact extraction with a focus on encrypted cryptocurrency wallets and messenger services [4].

A class of applications for which there has been little research as of today is password managers. A password manager is an application that holds login credentials and other sensitive information. Access to such data can be enormously valuable to an investigator, as it might not only provide insight into which services a suspect is using, but also providing access to these services. However, password managers are applications with high security requirements. Failure to capture relevant data on a running device immediately at the time of seizure makes it often impossible to access the encrypted data later.

Despite the obvious forensic value of password manager data and the fact that it has been shown that relevant data from password managers can be acquired through memory forensics [9], [10], the author was not able to find a comprehensive description of a forensic process on how to reliably extract such data. This gap shall be addressed in this thesis.

Research in this area might be able to provide investigators with means to access relevant data that is, as of today, not easily accessible. Such data might help in improving the investigative process and in solving criminal cases – therefore providing value for both law enforcement and society as a whole.

1.3 Research aim

To address the above-mentioned research gap, this thesis is guided by the following research aim: *Develop a process on how to extract forensically relevant data from password manager applications.*

This process shall consider previous work in the field of memory forensics and current best practices in digital forensics. More specifically, this thesis shall build upon the work done by Bang et al. with their "empirical framework for examiners to accessing password-protected resources for on-the-scene digital investigations" [4] and extend it to specifically include password manager applications.

1.4 Delimitations

Digital forensics is a vast field. It is therefore necessary to limit the scope of this thesis to do justice to the limited resources available.

– There are countless password manager applications on the market. This thesis only considers two of the most widely used: *Bitwarden* and *KeePass* (see chapter 4.1 for details about this

choice). This is to ensure a high level of practical relevance while keeping the effort within the boundaries of this thesis.

– Memory and application forensics vary widely between different operating systems (OS). For the same reason as stated above, this thesis can only consider the most widely used OS: *Microsoft Windows*.
– While this thesis aims to contribute results that can be applied in forensic investigations, the testing of the proposed process is limited to an artificial setting with defined scenarios. A real-world evaluation would bring significant ethical and organisational challenges and, therefore, would require more resources than are available.

2 Extended Background

2.1 Digital Forensics

2.1.1 Forensic Soundness

Simply put, forensic science can be defined as “the application of scientific methods to establish factual answers to legal problems” [1, p. 2]. A core principle of forensic science is forensic soundness. Performing forensics in a sound manner means that the authenticity and integrity of forensic artefacts can be validated at all times, and that results are reliable and repeatable [11, pp. 3–5].

Digital forensics “refers to forensic science applied to digital information” [1, p. 4]. To ensure forensic soundness, a digital forensics examiner would traditionally create a forensic copy of all seized storage media without altering the original medium. With larger and more distributed storage, however, this is no longer feasible. Therefore, there needs to be a balance between “the need to extract the most useful digital evidence as efficiently as possible, and the desire to acquire a pristine copy of all available data without altering anything in the process” [11, p. 3].

Furthermore, the acquisition of certain types of data – like, volatile memory – often requires an alteration of the system, as an application for securing memory must run in memory itself. To ensure forensic soundness under these circumstances, proper documentation is crucial. [11, pp. 3–5]. Forensic examiners therefore rely on standardised processes and procedures.

2.1.2 The Forensic Process

According to Flaglien [2], the digital forensics process can be divided into 5 phases: Identification, collection, examination, analysis, and presentation.

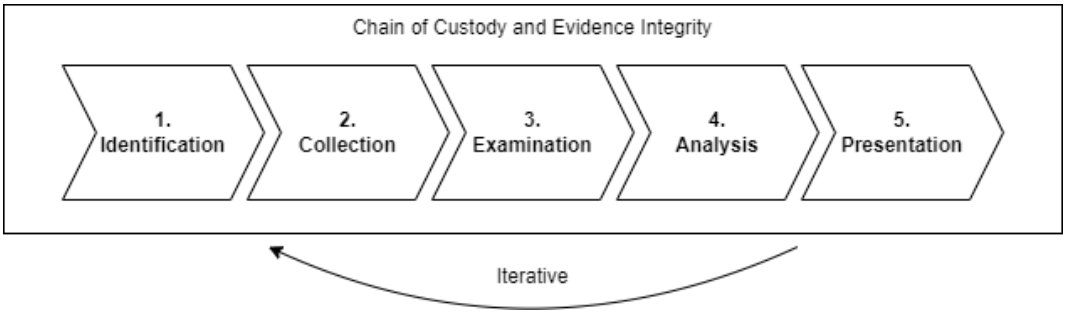

Figure 1: The digital forensics process (own illustration, based on Flaglien [2, p. 16])

For this thesis, the three first phases are most relevant. This is why they will be described in detail below while the other two are only briefly mentioned.

– *Identification*: The identification phase contains the tasks of detecting, recognizing, and determining the incident or crime to investigate [12]. An important part to successfully perform this phase is preparation. Law enforcement agencies need to have the right tools and processes – like the framework to acquire relevant artefacts that will be presented in a later chapter – in place before a crime happens. Also, staff – especially first responders – need to be trained to recognize devices with potentially relevant (volatile) digital evidence and handle them properly. Live systems might auto-lock (or run out of power) after a certain time, therefore only leaving a limited time frame to acquire data. The identification phase might also include planning the seizure of devices; for example, to try to capture them in an unlocked state.
– *Collection*: At the site of a crime, an examiner needs to collect "data from digital devices to make a digital copy using forensically sound methods and techniques" [2, p. 25]. This phase is also called data acquisition or extraction. An important concept in the collection phase is the order of volatility. As not all data has the same "lifespan", it is important to copy the most volatile data first. As mentioned in the previous chapter, it is impossible to collect all data without making changes to the system. The presence of standardised, approved, and repeatable procedures – like the mentioned framework – are crucial to ensure forensic soundness.
– *Examination*: The collected data requires "restructuring, parsing, and pre-processing of raw data to make it understandable for a forensic investigator in the upcoming analysis" [2, p. 33]. This is especially true for memory data: after copying, a memory dump is nothing more than a large chunk of unstructured, binary data. A forensic examiner needs to know what kind of data she is looking for and where to look for it. As different applications as well as different operating systems handle data distinctly, the task of extracting relevant data as potential evidence can only be achieved efficiently by means of semi-automated or automated tools that encapsulate knowledge gained from research in digital forensics.

In the *analysis* phase, an examiner connects the extracted artefacts to the case at hand. They will check which artefacts are relevant, how they connect to the investigated persons, and whether the evidence supports or refutes an investigation hypothesis. Lastly, in the *presentation* phase, the results from the analysis are shared with the relevant parties in the form of a report.

2.2 Memory Forensics

2.2.1 Development

The scientific community first gained interest in RAM analysis in the early 2000s. The first approaches to extract relevant data from memory dumps used simple text-based search tools like

grep and *strings* [3]. This method, also called unstructured analysis, comes with various limitations as it does not allow to search in complex data structures and requires an examiner to have a clear idea what she is searching for. Nowadays, this type of analysis can still be relevant when searching for known patterns, e.g., encryption keys.

With the development of a framework called *FATKit* [13], and memory analysis tools such as *Volatility*, memory analytics became more sophisticated. Through understanding how different operating systems organise memory, these tools are able to perform more a detailed, structured analysis [14].

While memory forensics was traditionally used to analyse malware or to gain insights after an incident, the ability to recover information that cannot be extracted through disk forensics soon gained examiners' interest. Especially the capability to access cleartext information that is encrypted when stored or transferred can be of high value during an investigation [3].

2.2.2 Application-specific Analysis

Memory analysis is a vast field with much ongoing research. The ways how operating systems handle memory are modified constantly and new security models make it more difficult to gain access to raw memory data. This leads to constant new challenges for memory forensics researchers and tool developers [15].

Processes and applications run by the user will store data in the so-called user space – outside the operating system kernel and usually isolated from other processes [16]. These application-specific memory areas can contain relevant data in connection with an investigation. Browser memory can contain submitted POST data (e.g., passwords), email or messenger applications memory can contain cleartext messages that are not stored (or stored encrypted) on disk, and memory from an encryption software like *Veracrypt* can contain passwords or encryption keys. However, for a long time, such information needed to be extracted manually due to the lack of automated solutions [3]. Furthermore, data captured from memory (e.g., encryption keys) often needs to be combined with data from permanent storage (e.g., an encrypted database) to gain useful information. This requires an in-depth knowledge of the workings of the respective applications.

In the last few years, research has developed approaches to make the analysis of application-specific data more efficient. However, as every application works differently, and as applications change over time, more research is needed in this field [3], [6], [17].

2.3 Frameworks for Digital Forensics

2.3.1 Combining the Pieces

While the increasing amount of research in the field of live and memory forensics produces much useful knowledge and many effective tools for forensic analysts, the scattered nature of the research makes it hard to apply them under the limited time and resources available in a crime investigation. The forensic process relies on standardisation and structure.

Recent work, therefore, increasingly combines different research results into comprehensive processes or frameworks that can be applied in an investigation. For example, Kazim et al. [6] presented a workflow to recover chat messages and encryption keys from seized devices, Fernández-Álvarez and Rodríguez [17] demonstrated a three-step process to extract and analyse relevant artefacts from the *Telegram* desktop application on *Windows*, and Mistry and Dahiya [18] developed a framework for analysing network-based artefacts from volatile memory.

2.3.2 Vision

In 2022, Bang et al. [4] proposed *Vision*, an "empirical framework for examiners to accessing password-protected resources for on-the-scene digital investigations". The framework is a comprehensive process to extract relevant artefacts from memory and combining these findings with further artefacts identified in disk storage. In its presented form, *Vision* focusses on the acquisition of artefacts related to cryptocurrencies, device lock and encryption keys, and secure messenger and social media data. The authors, however, intended *Vision* to be an open, modular framework that allows adding new analysis targets.

The framework's workflow is divided into four steps:

- During *pre-definition*, application-specific variables are defined. This includes typical filenames, paths, and process names; patterns of relevant strings (e.g., length of encryption keys); and the sub-procedures that shall be applied to analyse memory and files related to the application.
- During *discovery*, the system is searched for relevant application data, and relevant artefacts within the application data is identified and extracted (based on the definition from the previous step).
- During *analysis*, the extracted data is combined (if necessary) and verified (e.g., it is checked whether an extracted encryption key can be used to decrypt an encrypted file).
- During *post-processing*, the results from the analysis step are summarised and made available to the examiner.

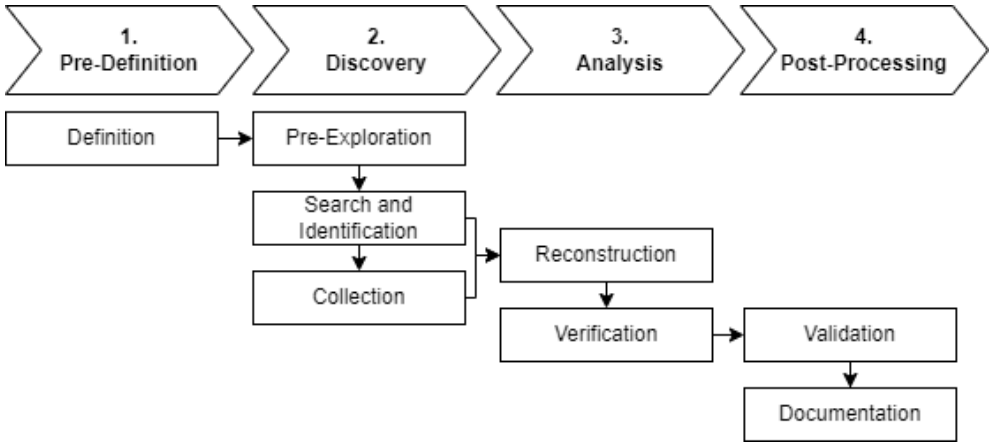

Figure 2: Vision Framework workflow (own illustration, based on Bang et al. [4])

This thesis will build upon the framework and its structure as proposed by Bang et al. The modular structure of the framework shall be leveraged to introduce the ability to extract and analyse relevant artefacts related to password managers.

2.4 Password Managers

2.4.1 Passwords and their Weaknesses

Passwords are still the dominant method of authentication when using digital services despite it being known that password authentication has some severe shortcomings. The two most common problems are users choosing weak passwords, and users reusing the same password for different services [19].

Both problems can be mitigated using a password manager. A password manager is an application that randomly creates and stores strong, unique passwords for every service. The database containing all credentials is securely stored using modern encryption algorithms. The encryption is done based on a single master password that a user must provide every time she unlocks the database.

2.4.2 The Role of Password Managers

It is no surprise that security experts recommend the use of password managers [20]. With data breaches continuously in news, and the associated higher awareness among the general population, the use of password managers is on the rise. In a recent survey, 34 % of participants claimed that they currently use a password manager [21]. It can, therefore, be expected, that more and more devices seized during a criminal investigation will have password manager data on them.

There are many different applications on the market. Password managers can either be integrated in an operating system (like *Apple Keychain*) or browser (e.g., *Google Password Manager*, *Firefox Password Manager*), or they can be stand-alone applications. The latter are mostly available in a restricted free and a premium version, and most modern password managers are cloud-based software-as-a-service (SaaS) applications. This allows synchronisation of credentials over multiple devices while guaranteeing security through strong end-to-end encryption. While no comprehensive usage statistics about the different products are available, tech magazines periodically publish articles about the "best password managers", giving insight in the most popular products. Today, these include, among others, *1Password*, *Bitwarden*, *Dashlane*, *LastPass*[1], *Keeper*, and *KeePassXC* [23], [24]. A more comprehensive overview about different products can be found in chapter 4.

2.4.3 Forensic Extraction of Artefacts

As a vulnerability in a password manager has the potential to expose all login credentials of a user, security is of utmost importance. Manufacturers are aware of their responsibility and implement high security standards. Safety measures are also regularly scrutinised by researchers [25]. From a forensic point of view, it must therefore be assumed that a locked and non-running password manager does not allow access to stored credentials without the cooperation of the owner (that is, if the master password is not known).

However, whenever a user enters his master password to access stored credentials (e.g., to log in to a service), both the cleartext password as well as the encryption keys derived from it must be processed in memory. This leaves an opportunity for forensic examiners to gather relevant data if a device can be seized while running a password manager in a certain state.

Sabev and Petrov [26] demonstrated that it is possible to extract the master password from *Bitwarden* and *Keeper* on Android from memory. The security company Paradox Infosec [10] demonstrated that the master password of a running *KeePass* instance can be extracted from a remotely controlled low-privilege user account in *Windows 10* with the use of an open-source script called *KeeThief* [27]. And Nusbaum and Perez [9] compared memory extractions of artefacts related to the *LastPass Chrome* browser extension during different stages of application use.

[1] LastPass was affected by a severe data breach in 2022. While stored passwords are protected through end-to-end encryption, a third party gained access to personal user data and unencrypted metadata [22]. Some media outlets removed LastPass from their list of recommended password managers as a reaction.

2.5 Summary

To ensure evidence integrity and forensic soundness during an investigation, it is important that evidence is handled and documented correctly form the very beginning. To accomplish this, standardised processes and procedures are the norm in the field of forensics. With the rise of encryption, the importance of live forensics performed at the time of seizure increased significantly. For this reason, research in the field of memory forensics has grown strongly in recent years.

However, research in memory forensics is often focussed on specialised issues and applications, and the results are presented in isolation. While first approaches are visible to combine different results in a forensic framework, there are still many pieces that need to be added. This thesis aims to contribute to this cause by (a) systematically identifying forensically relevant artefacts related to password manager applications, and (b) proposing a process to extract and analyse these artefacts based on the existing *Vision* framework.

3 Methodology

3.1 Introduction

The aim of this thesis is to propose a way on how to extract forensically relevant data from password manager applications. As this research aim is not purely empirical, but instead has the creation of a new artefact as its goal, this thesis will follow a design science approach. More specifically, it will follow the design science framework as suggested by Johannesson and Perjons [28]. According to the authors, their approach shall help "developing artefacts that can help people fulfil their needs, overcome their problems, and grasp new opportunities" [28, p. 1].

Design science is not just about developing an artefact, it is also about creating knowledge around it. To do this, design science often combines different research strategies and methods. Johannesson and Perjons structured the process into five distinct steps that shall also guide the research done in this thesis. The steps and the applied research strategies and methods are discussed in the following chapter.

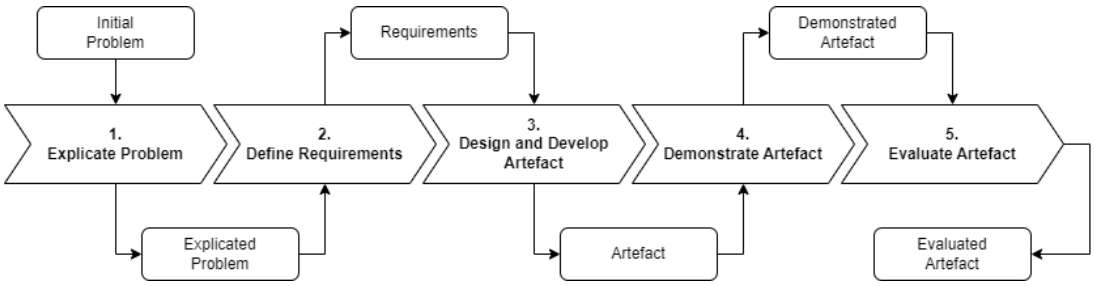

Figure 3: The design science process (own illustration, based on Johannesson and Perjons [28, p. 77])

3.2 Design Science

3.2.1 Explicate Problem

The first step in the design science process is to "formulate the initial problem precisely, justify its importance, and investigate its underlying causes" [28, p. 91]. In this thesis, this is done through literature review. The author considers the problem as already well-known and sufficiently described in past work, so that further in-depth research would not add significantly to the understanding. The problem is described in chapter 1.2 with the relevant background knowledge summarized in chapter 2.

An alternative research strategy for this step would be to perform a survey and collect insights from stakeholders (e.g., investigators) through interviews. However, this approach would consume much of the available resources for this thesis while, probably, not providing many new insights. It was therefore decided to focus the available resources on the later parts of the process.

3.2.2 Define Requirements

The second step in the design science process is to "identify and outline an artefact that can address the explicated problem and to elicit requirements on that artefact" [28, p. 103]. The requirement definition can be viewed as an extended problem explication. This thesis is building upon an existing forensic extraction framework (the *Vision* framework as described in chapter 2.3) and extending it with an instantiation that can be used to extract relevant information from password managers. The requirements are therefore mainly given by the framework structure and therefore taken from literature. They are listed in chapter 4.1.

Viable alternative research strategies for this step could be a survey (interviews with stakeholders), or a case study (applying the existing framework in a real-world scenario and identify weaknesses to further improve it). While such an in-depth definition of requirements might result in a better-grounded artefact, the limited resources of this thesis do not allow following any of these strategies. Furthermore, the existing knowledge from previous research is deemed sufficient to obtain a meaningful outcome.

3.2.3 Design and Develop Artefact

In the third step of the design science process, the artefact is created. During this step, research strategies and methods can be used, but they are not as crucial as during the other steps. "Any approach for generating solutions is admissible, as long as it works" [28, p. 25].

To create a suitable artefact, the following steps are performed:

1. A reasonable selection of two out of all available password manager applications is made to base the artefact on. Gaining a deep understanding on how a software works is a time-intensive task. It is therefore appropriate to use the limited resources available in this thesis to thoroughly investigate a limited number of programs than to shallowly analyse a larger set. Despite this limitation, the resulting artefact is still expected to be of broader relevance. It is reasonable to assume that the inner workings do not differ much among different password manager applications. Their main task is to encrypt and safely store credentials. With only a limited number of encryption algorithms that are appropriate for this task, we can assume that the findings from the smaller sample will have relevance for other products as well.

2. The selected password managers are analysed to identify relevant forensic information that can be extracted from them. This is done using two data collection methods: document research and observation. Software with a focus on security often comes with publicly available documentation about their security mechanisms (e.g., published in a so-called whitepaper). Sometimes, software might even be fully open source, allowing access to the complete source code. These documents will be used to gain an understanding about the inner workings of the examined password managers. This understanding will be supplemented by additional data gained through observing the program behaviour in an experimental setting. After installing the software in a dedicated virtual machine and performing regular user actions, the state of the system – both files and memory – is recorded at different points in time. The application files and memory dumps are then searched for relevant forensic artefacts. As the observation is performed in a controlled setting, both input and output are known to the researcher. The search is therefore conducted by a string search for known input strings in both the file system and the memory dumps.
3. The collected data is qualitatively analysed and used to create an instantiation of the forensic method defined in the *Vision* framework that can identify and extract forensically relevant information from password managers. This instantiation includes processes for the identification and extraction of relevant artefacts from application files and memory, the processing and validation of these artefacts, and the creation of a report. Data collection and analysis will be iteratively performed until enough information is available to fill every step of the framework.

An alternative data collection method for this step could have been interviews with developers of password manager applications. As creators of the software, they would be able to give an in-depth insight into the applications' inner workings. However, access to such persons might be difficult, and interviews are time-consuming and therefore problematic regarding the limited resources available. Given the extensive amount of openly available information, the chosen approach is much more efficient in gaining the required data to base the artefact on.

3.2.4 Demonstrate Artefact

The fourth step of the design science process is to demonstrate the use of the artefact to show that it is suitable to solve the problem. As "a demonstration can be seen as a weak form of evaluation" [28, p. 133], this thesis closely connects this step with the evaluation step.

The artefact is applied in an ex-post, artificial experimental setting [28, p. 139]. The goal of this proof-of-concept is to show the practical relevance to a forensic investigation. To do this, a testing environment within a virtual machine (VM) is set up and realistic scenarios are defined. The approach of testing a similar artefact against scenarios has already been used by Bang et al. in

their initial *Vision* paper [4], as well as in other research papers on memory forensics (e.g., [17], [26]). The aim of this simulation is to perform an extraction of password manager data as it could look like in a real investigation.

While a laboratory experiment allows testing of the extraction process in a controlled and protected environment, it comes with certain downsides [29, pp. 66–78]. Most notably, a laboratory experiment can never completely simulate the real world. To fully evaluate the usefulness of the artefact, a field experiment would therefore be more meaningful. However, such an experiment would require taking part in a real-life investigation, which is out of scope for this thesis. Furthermore, applying a never-tested artefact in a real case would bring risks to the forensic soundness and therefore the success of the investigation. This would not be justifiable from an ethical point of view.

3.2.5 Evaluate Artefact

The fifth and last step in the design science process is to "determine to what extent an artefact is effective for solving the problem for which it has been proposed" [28, p. 137]. This is done through a summative evaluation using the experimental setting described in the demonstration step.

As the demonstration is done in a controlled environment, both inputs (data that is stored in the password manager) and outputs (data that could be extracted using the artefact) are available to the researcher. The evaluation therefore measures how much of all possible relevant data is extracted when applying the artefact. This way, the effectiveness of the artefact can be described. Furthermore, the final artefact is compared to the requirements stated in 4.1.

An alternative evaluation strategy would be to perform a naturalistic evaluation using a case study. However, for the reasons as described under the demonstration step, any application of an untested artefact in a real-life investigation is neither responsible nor ethically justifiable.

3.3 Ethical Considerations

The present research is carried out in compliance with ethical principles. Denscombe [29, pp. 306–324] defines four key principles for ethical research: 1) protect participants interests, 2) ensure voluntary participation and consent, 3) avoid deception and operate with scientific integrity, 4) comply with the law.

The first three principles are of negligible importance for this thesis as no participants are involved. Data collection involves documentation and open-source code, both of which are public information without a risk to cause harm to individuals or organisations.

Two steps are taken to ensure compliance with the law. Firstly, no personal information is processed, and only publicly available information is used. Secondly, a deep examination of software can conflict with intellectual property rights. To avoid such conflict, no intrusive techniques such as reverse engineering are used.

Information security research often comes with another ethical dimension. Although the aim of this thesis is to support law enforcement in conducting more effective investigations, the results can also have the potential to be misused by malicious actors. In the present work, the opportunities outweigh the risks for two reasons: Firstly, this thesis concerns digital forensics carried out with physical access to running devices in, ideally, a specific state. While law enforcement has legal means to gain such access, this is much harder for a malicious actor. Secondly, this thesis is concerned with how information can be extracted from a software that is used as intended by the vendor. It does not look for vulnerabilities nor unintended use cases. Should a vulnerability accidentally be discovered, so will it be disclosed responsibly to the vendor.

4 Password Manager Forensics

4.1 Artefact Requirements

The artefact[2] to be developed in this thesis is an instantiation of a forensic extraction method defined in the *Vision* framework. Requirements are therefore derived from existing literature on forensic frameworks (as described in chapter 2.3). The main goal of the artefact is to extract forensically relevant password manager data from a file and memory image of a seized device.

Structurally, the artefact shall follow the structure of the *Vision* framework (as described in chapter 2.3.2). Functionally, the artefact shall (1) describe and define forensically relevant artefacts; (2) define methods and functions to extract said forensic artefacts; (3) provide said methods and functions; and (4) present the extracted information in a compact and clear way. Furthermore, the artefact shall maintain central attributes of *Vision*: It shall be (5) modular and flexible, to allow adding more password manager applications, and adapting it to future changes in the applications; and it shall be (6) comprehensible and accessible through open and thorough documentation.

4.2 Choice of Sample

To select a suitable sample of password managers, a list with the most popular applications was created, and the different applications were compared. As no independent and reliable information on usage numbers of different password managers was available, popularity was assumed if a password manager was recommended by renowned magazines. Therefore, all password manager applications that have been recommended in recent articles from *Forbes* [24], *Wired* [30], and *CNET* [23] were examined in further detail. The full overview is available in appendix B.

The three articles recommend a total of eleven password managers. Five applications were recommended in all three articles: *1Password*, *Bitwarden*, *Dashlane*, *KeePass*[3], and *Keeper*. Two further applications – *LastPass* and *Nordpass Password Manager* – have been recommended by two magazines. Four password managers appear in only one article.

[2] The term *artefact* describes both the developed result in design science, and a piece of forensic evidence. As *artefact* is the most common term used in both disciplines, this thesis also uses it for both meanings. While this might seem confusing at times, context should always allow to determine the current meaning.
[3] *KeePass* is an open-source project actively developed by a large community. Some of the articles recommended *KeePassXC*, the most popular fork of the original *KeePass* application. In the password manager comparison, the "KeePass ecosystem" has been treated like a single application, therefore including community-developed parts such as mobile apps (*KeePass2Android*, *Strongbox*), or browser extensions under the name *KeePass*.

For the reasons outlined in chapter 3.2.3, this thesis will focus on a sample of two password manager applications. For this, *Bitwarden* and *KeePass* have been selected from the list for three reasons: (1) Both applications have been recommended in all three reviewed articles and can therefore be considered among the most popular password managers; (2) As the only applications from the list, these two are both fully open-source and with a comprehensive documentation of the security models – therefore providing suitable information for the definition of relevant artefacts; (3) The selection allows to examine two different type of password managers: While *Bitwarden* is a cloud-based software-as-a-service (SaaS) offer, *KeePass* is a local stand-alone software.

To observe the behaviour of the applications and therefore gain a deeper insight into relevant artefacts, the *Windows* stand-alone applications for both password managers will be examined[4]. The limited scope of this thesis does not allow to analyse all the many clients; therefore, the most used operating system shall be in the focus. As the core security model of a password manager is platform-independent, this approach allows to gain insights that can easily be adapted to other platforms while at the same time performing observations in a scenario that is one of the most likely ones to appear in a real-life investigation. Details about the examined application versions can be found in appendix C.

4.3 Relevant Forensic Artefacts

4.3.1 Bitwarden

All sensitive data (especially login credentials) is saved in what most password managers call a vault. *Bitwarden's* vault data is encrypted using AES-CBC with a 256-bit key. This key is generated on account creation and encrypted (again, using AES-256) with a key that is derived from a user's master password and a user's account email address. The cleartext vault key (called symmetric key in the figure below), as well as decrypted vault data is only available in memory. The encrypted vault data and vault key are stored on disk (and transferred to *Bitwarden*'s server).

[4] The "*KeePass* ecosystem" has yielded several *Windows* forks of the original *KeePass* application. This thesis uses the original *KeePass* application for *Windows*.

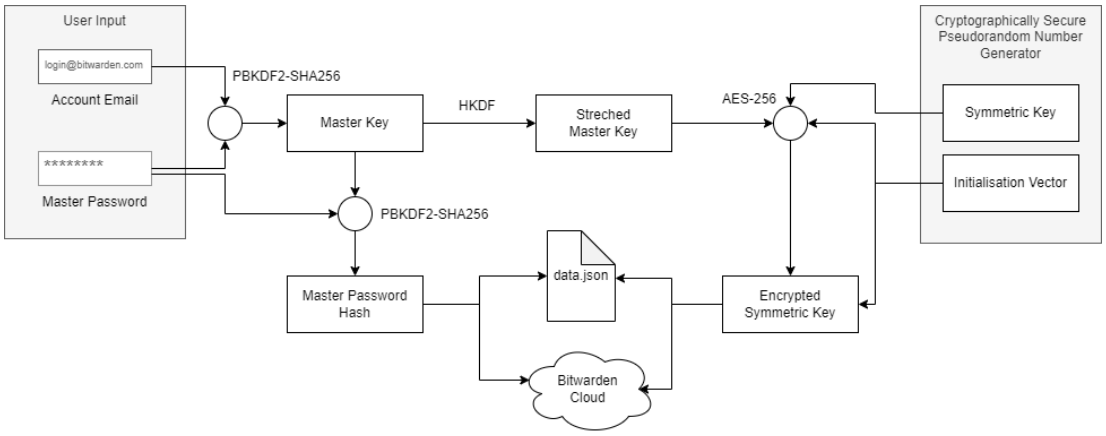

Figure 4: Cryptography overview of Bitwarden (own illustration, based on the Bitwarden Security Whitepaper [31])

Windows application files can be stored in three different locations, depending on how the application was installed (stand-alone installer, from *Windows Store*, or as a portable app) [32].

Bitwarden allows users to lock the local copy of their vault with a PIN code for convenience. When activating this function, users decide whether they will need to enter the master password after restarting the application, or if they always want to unlock their vault with a PIN only. In the latter case, a second master key is derived from the PIN (using the same cryptography as shown above). This key is then used to encrypt the master key and store it in a file. In such a setting, a weak PIN code might open an attack surface for brute-forcing.

As no data is transferred to the server that cannot also be secured from a seized device, and as all transfers between a *Bitwarden* application and the server are secured with *TLS*, network artefacts are not considered in this thesis. For the same reason, the master password hash is excluded as it is only used to authenticate the app to the server and does not provide forensic value.

Table 1: Summary of main artefacts: Bitwarden

Artefact	Description	Memory	Storage
Master Password	A user-provided passphrase to unlock the vault	x	
User Account Email	Email address used to sign-up for the account	x	x
Master Key	Derived from the master password through PBKDF2 (using the user account email as salt)		
- Encrypted			(x)[5]
- Unencrypted		x	
- No. of PBKDF iterations			x
Vault Key (Symmetric Key)	256-bit key used to encrypt the vault data		
- Encrypted			x
- Unencrypted		x	
Vault Data	User-stored information (e.g., login credentials) as well as account metadata		
- Encrypted			x
- Unencrypted		x	(x)[6]

File-based artefacts can be extracted from the *Bitwarden* application directory. All forensically interesting information is located in a file called *data.json*. While most data inside the *data.json* file is encrypted as what *Bitwarden* calls a *Cipher String*, some information like the time of last cloud synchronisation, or password generator settings are not.

Memory-based artefacts can be extracted from a memory dump that is created as soon as possible after device seizure. A string search for known vault data in memory dumps taken at different points in time revealed that the availability of the different artefacts depends on the state of the application. Four states were observed: (1) app closed, and user signed out; (2) app closed, and user signed in; (3) app open but with a locked vault; (4) app open, and vault unlocked.

[5] If a user has set a PIN code to unlock the vault and disabled the option "Lock with master password on restart", the master key is stored in the *data.json* file, encrypted with a key derived from the PIN code.
[6] Some metadata that can be useful in a forensic investigation is stored in cleartext. This includes, for example, creation and edit dates of vault items, the last sync date, and password generator settings.

Table 2: Availability of Bitwarden artefacts depending on application state

Artefact	Closed (signed out)	Closed (signed in)	Vault locked	Vault unlocked
Master Password	(x)[7]	(x)	(x)	x
User Account Email	(x)[8]	x	x	x
Master Key				
- Encrypted		(x)[10]	(x)	(x)
- Unencrypted				x
- PBKDF iterations	x[9]	x	x	x
Vault Key (Symmetric Key)				
- Encrypted		x	x	x
- Unencrypted				x
Vault Data				
- Encrypted		x	x	x
- Unencrypted		(x)[11]	(x)[12]	x

A detailed description of the above-mentioned artefacts can be found in appendix E. The process and structure of this definition is described in chapter 4.4.1.

4.3.2 KeePass

KeePass is a local password manager without cloud synchronisation. All vault data is stored in a *KeePass* database file (.kdbx). When a user first opens the application, a new database file is created, and the user can choose where to save it (default: the users Documents folder). The database is encrypted using a master password and/or a key file (.keyx)[13]. *KeePass* supports two encryption algorithms (AES-256 and ChaCha20) while AES-256 is the default setting. The vault encryption key is derived from the master password and/or the key file by first hashing this data with SHA256 and then applying a key derivation function. By default, *KeePass* uses an AES-based key derivation function with 60'000 iterations, however, users can opt to use *Argon2* instead. Furthermore, the *KeePass* database is compressed with *GZip* by default.

[7] According to *Bitwarden*'s security whitepaper, "the Master Password is cleared from memory after usage" after the app has been closed or the vault locked [31]. However, in the memory dump shortly taken after logging out and/or closing the app, the master password was still accessible.
[8] If the user ticked the box "remember email" at the last sign-in.
[9] The number of PBKDF iterations for a given account email address can also obtained through an API call to https://vault.bitwarden.com/identity/accounts/prelogin
[10] If a user has set a PIN code to unlock the vault and disabled the option "Lock with master password on restart", the master key is stored in the *data.json* file, encrypted with a key derived from the PIN code.
[11] Some metadata that can be useful in a forensic investigation is stored in cleartext.
[12] *Bitwarden* claims that "after a certain time frame of inactivity on lock screen, we reload the application processes to make sure that any leftover managed memory addresses are also purged" [31]. In the observed cases, some unencrypted vault data was available in memory a short time after locking the vault, other was not.
[13] A third option for "expert users" is to leverage Windows User Account keys to encrypt the KeePass database. This option is not investigated further in this thesis as its use is discouraged by KeePass [33].

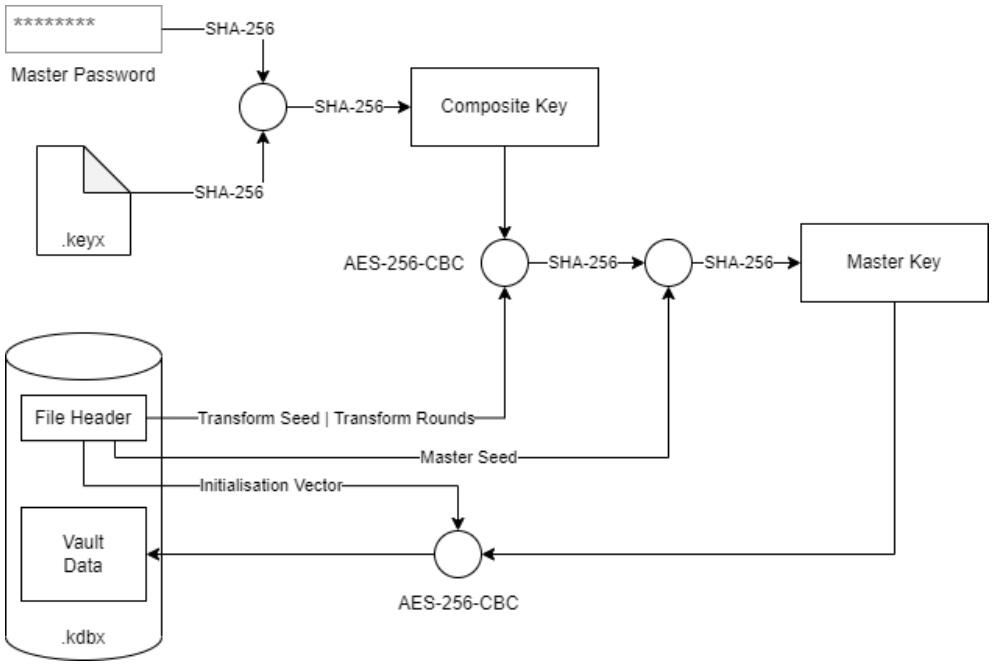

Figure 5: Cryptography overview of KeePass (own illustration, based on Velde [34])

As all user data is stored in the database file, not much information is available from the application directory. However, the *Windows* installer version of *KeePass* creates a configuration file (*KeePass.config.xml*) that stores the last used database and key files, as well as password generator settings.

Table 3: Summary of main artefacts: KeePass

Artefact	Description	Memory	Storage
Master Password	A user-provided passphrase to unlock the vault	x	
Key File	Alternative or additional key to the master password		x
Master Key	256-bit key used to encrypt the vault data. Derived from the master password and/or key file through AES-iterations		
- Unencrypted key		x	
- Transform seed			x
- No. of iterations			x
Vault Data	User-stored information (e.g., login credentials) as well as account metadata		
- Encrypted			x
- Unencrypted		x	

As *KeePass* is not a cloud-synchronised application and, therefore, does not rely on user accounts, only three states have been analysed: (1) app closed; (2) app open, but vault locked; (3) app open, and vault unlocked.

Table 4: Availability of KeePass artefacts depending on application state

Artefact	App closed	Vault locked	Vault unlocked
Master Password			(x)[14]
Key File	x	x	x
Master Key			
- Unencrypted key			
- Transform seed	x	x	x
- No. of iterations	x	x	x
Vault Data			
- Encrypted	x	x	x
- Unencrypted		(x)[15]	x

A detailed description of the above-mentioned artefacts can be found in appendix E. The process and structure of this definition is described in chapter 4.4.1.

4.4 Forensic Process (Artefact)

This chapter describes the developed artefact to extract the above-mentioned forensically relevant information from password manager applications. Like *Vision*, it consists of several definition files and different modules written in *Python*. The artefact is called *Password Manager Forensics*, or *PMF* for short.

4.4.1 Pre-Definition

As a basis for the next steps in the process, the information described in the previous chapter was prepared in a structured and, where appropriate, machine-readable format. The pre-definition files for the process proposed in this thesis follow the structure of Bang et al. [4]: Firstly, common application paths are defined to verify that an application of interest is installed, and relevant files are defined so that they can be extracted from the file system. As described in 3.2.3, paths and file names were elicited from the available documentation and through observing the software in an experimental set-up. Like in the *Vision* framework, *JSON* was the format of choice to write the file artefact pre-definition file.

[14] According to *KeePass*' security whitepaper, sensitive data is stored encrypted in the process memory [35]. However, in some of the analysed memory dumps, the master password was available in cleartext.
[15] According to *KeePass*' security whitepaper, all security-critical memory is erased when it is not needed anymore [35]. However, in the memory dump shortly taken after locking the app, cleartext vault data was still accessible.

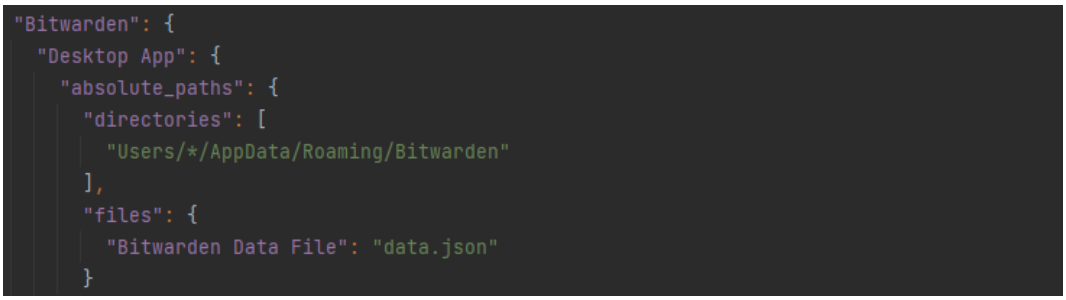

Figure 6: Example definition of relevant application paths and names for Bitwarden

Secondly, the relevant forensic artefacts as well as their structure within the identified files and within memory are defined to allow searching for them. To do this, the collected information about the applications (as described in 3.2.3) is aggregated and analysed for patterns. For example, different memory dumps that all include the (known) cleartext master password are compared to find the byte sequences around the password that stay constant. This byte sequences can then be used to look for unknown master passwords in future memory dumps (see also 4.4.2).

As these artefacts are dependent on the application, one file per password manager is created. This documentation is the basis for processing the files during the discovery and analysis phase. For easy and platform independent readability, the documentation is written using *Markdown* syntax [36].

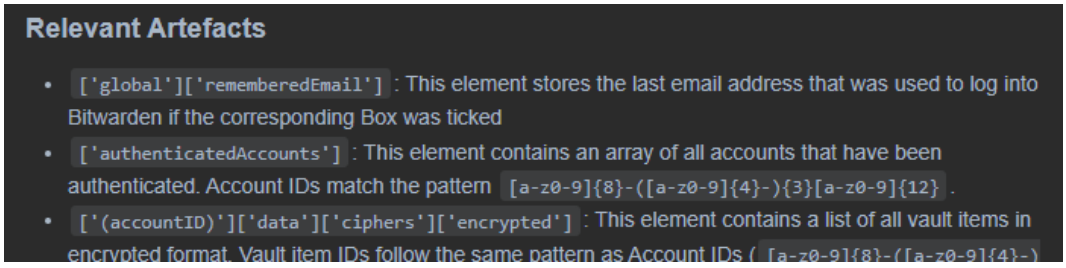

Figure 7: Example definition of relevant artefacts and their structure

Thirdly, the functions and modules necessary to do the extraction according to the defined patterns are defined. As in the search pattern definition, one *Markdown* file per password manager is created to document the distinct functions and methods to extract and analyse the defined artefacts.

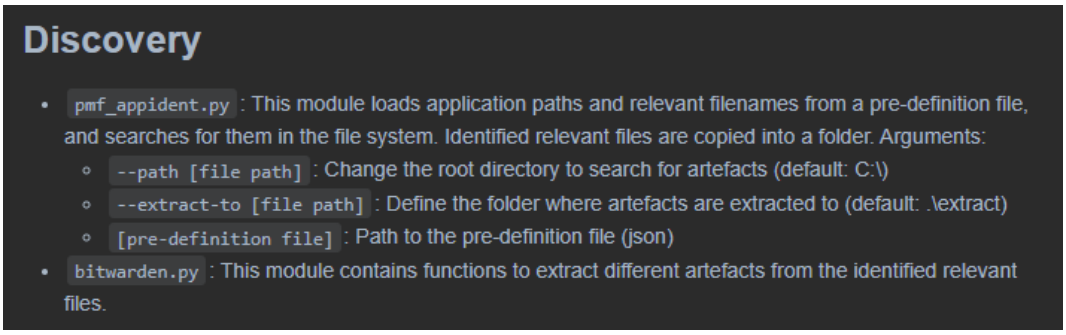

Figure 8: Example definition of modules and functions

Details about the mentioned functions and modules are explained in the following chapters. The full pre-definition documentation can be found in appendix E. The full code base including all pre-definition files is further available on *GitHub*[16].

4.4.2 Discovery

The discovery step consists of three parts: Firstly, the file system is scanned to check if a password manager application is installed on the device (or within a mounted file image). If this is the case, relevant files (as defined in the pre-definition) are copied to a separate folder on the investigator's machine. Secondly, the extracted files are scanned for the forensic artefacts that have been defined and described under pre-definition. Thirdly, a memory dump is also scanned for forensic artefacts (again, looking for structures as defined in the pre-definition files).

To perform these tasks, two *Python* scripts have been developed. The *PMF Application Identifier* (*pmf_appident.py*) searches for installed password manager applications according to the pre-definition data in *pmf_apps.json*, and makes a copy of relevant files.

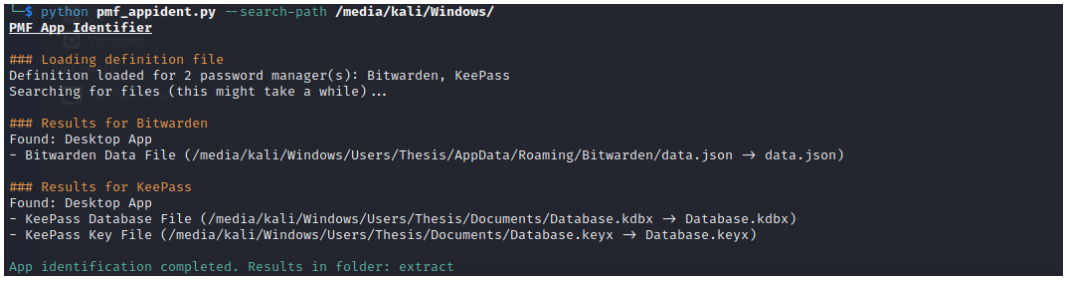

Figure 9: Example output from PMF Application Identifier

The extracted files can then be scanned for relevant artefacts using *PMF Extractor* (*pmf_extractor.py*). As the data structure is different for every password manager, a separate *Python* module has been created for every password manager, containing functions to extract the

[16] https://github.com/shaehni/password-manager-forensics

respective forensic artefacts as described in the pre-definition files. *PMF Extractor* further scans a memory dump for artefacts and extracts them. This is done using *Yara* rules [37]. While *Yara* usually is used to find indicators of malware, it is an efficient tool to scan large files for patterns, therefore making it ideal to search through memory dumps. As creating memory dumps is a standard functionality in most forensic tools, a function to dump memory is not part of the present artefact.

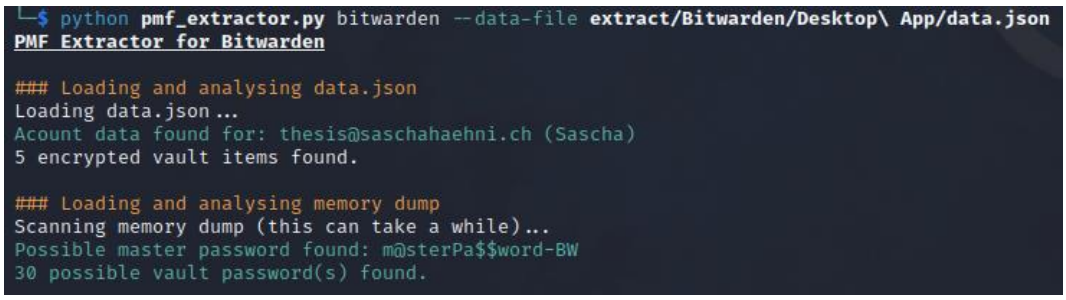

Figure 10: Example output from PMF Extractor during the discovery step

4.4.3 Analysis

The analysis step consists of two parts: reconstruction (making sense of the extracted information), and validation (checking if the reconstructed data is valid). With regards to password managers, reconstruction includes two actions: (1) checking if enough information is available to decrypt the vault, and (2) decrypt the vault if this is the case. Both examined password managers use message authentication codes (MAC) to ensure data integrity. *PMF* leverages this to validate the decrypted data. Therefore, both reconstruction and validation are done jointly. This step also allows to brute-force passwords or PIN codes. The analysis step is performed directly and automatically after the discovery step by *PMF Extractor*.

Figure 11: Example output from PMF Extractor during the analysis step

4.4.4 Post-Processing

In the last step, post-processing, the relevant information is converted into a document that is useful for an investigator. To do this, the *PMF Reporter* module (*pmf_reporter.py*) has been created. The module contains a report object that is instantiated at the beginning of the analysis, and continuously updated with relevant results as they are detected. For the end of the analysis, the module contains a function that exports all added results into a formatted text file.

A full example output of *PMF Reporter* can be found in annex F.

```
# ######
# Cryptography
# ######

Master Key: PkfJcJf/v21bHQDn0KRSXTAWAVG13yNUosXiWupSq6s=
Master Password: m@sterPa$$word-BW

Symmetric Vault Key:
- Encryption Key: jVZq5CM8k+SdtqYGHvukNbv+3wRaOM3zw1lfOeF5uY0=
- MAC: LX0TK1Yo1lE877kRmxdvD+EoRJI2HJsroIeYTVZEI48=


# ######
# Vault Data
# ######

Test-Login2:
- User: test.user@gmail.com
- Password: iS8rT5SNJXRnBK
- Notes: This is another note.
```

Figure 12: Example output of PMF Reporter

4.4.5 Artefact Overview

The following figure visualizes the structure and the different elements of *Password Manager Forensics (PMF)*. An overview of all files of the artefact can be found in appendix D.

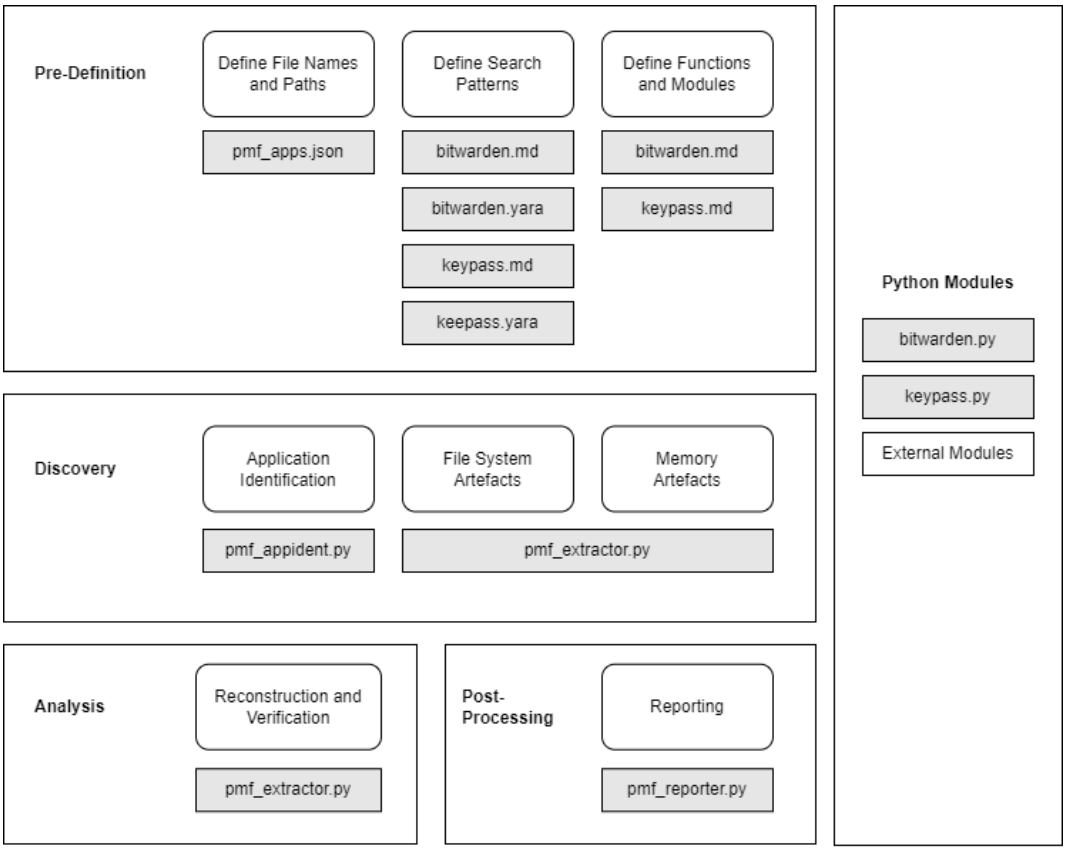

Figure 13: Structure of the artefact

4.4.6 Fulfilment of Requirements

In 4.1, six requirements for the artefact were defined. In this chapter, the final artefact is qualitatively evaluated against these requirements:

- The first and second requirement stated the artefact shall describe and define forensically relevant artefacts, and to define methods and functions to extract said artefacts. This requirement is fulfilled by the pre-definition part: Relevant file names, paths, and memory artefacts, as well as search patterns, are defined in detail for both password managers, and functions to extract artefacts are documented.
- The third and fourth requirement stated that the artefact shall provide above-stated methods and present extracted information in a clear and compact way. This requirement is fulfilled through the developed modules *PMF Application Identifier*, *PMF Extractor,* and *PMF Reporter*.
- Requirements five and six stated that the artefact shall be modular, flexible, accessible, and comprehensible. While developing the artefact, emphasis was placed on a modular structure that allows for an easy expansion with modules for other password managers, without the need to modify the core architecture. The artefact is thoroughly documented, both in this thesis document as well as in the pre-definition files (appendix E). By publishing the complete source code of *Password Manager Forensics (PMF),* the artefact is easily accessible to other scientists.

5 Proof of Concept

5.1 Scenarios

As the presented artefact is built upon the *Vision* framework, the structure of the evaluation scenarios was adapted from the paper by Bang et al. [4]. Consequently, three scenarios are defined to test the applicability of the developed artefact in a forensic investigation. All scenarios shall resemble a possible real-life case. To achieve this, the author took inspiration from real case examples described by Casey [11]. Furthermore, desirable outcomes (goals) are defined for each scenario that will act as a baseline for the evaluation. This has been shown to be a viable evaluation method for similar artefacts in the papers of Fernández-Álvarez and Rodríguez [17], and Sabev and Petrov [26]. To make the goals suitable for evaluating the presented artefact, they have been chosen in such a way that all the described functions of the artefact are addressed and, therefore, the artefact can be tested in its entirety.

5.1.1 Scenario 1: Locked Bitwarden Vault Items

During a search and seizure related to a possible fraud case, a *Windows* laptop could be secured. The investigators found both the operating system and a *Bitwarden* password manager application unlocked. However, investigators were not able to access the suspects *Gmail* and *Facebook* login credentials through the password manager's GUI as an additional security option was activated for these items: access requires re-entering the master password. Neither were the investigators able to export the whole vault database as this option requires re-entering the master password as well. Forensic experts created both a complete memory dump as well as a disk image for further examination.

Goal: Gain cleartext access to the login credentials to the suspect's *Gmail* and *Facebook* accounts. Access to these accounts could prove that fraudulent messages have indeed been sent by the suspect.

5.1.2 Scenario 2: Locked KeePass Database

The police conducted a house search in connection with a drug investigation. Among other devices, a *Windows* laptop believed to contain private keys to a suspect's *Bitcoin* wallet was seized. Furthermore, investigators found a notebook with what appears to be a list of passwords. None of the passwords granted access to the computer, however. The laptop was turned off, but the hard drive was not encrypted. Forensic experts were, therefore, able to create a disk image. Forensic software used by the examiners listed *KeePass* among the installed applications.

However, examiners were not able to identify relevant files belonging to *KeePass*, nor retrieve vault data.

Goal: Gain cleartext access to a *Bitcoin* private key stored in a *KeePass* database file hidden in the file system.

5.1.3 Scenario 3: Locked Bitwarden Account with PIN

During an investigation related to intellectual property infringement, a *Windows* laptop was seized while the device was running, and while the user was logged into the operating systems. From a desktop shortcut icon, investigators found that *Bitwarden* was installed on the machine. However, the application was not running. Examiners dumped the memory and created a disk image of the only hard drive for later analysis. In the lab, examiners found several encrypted *VeraCrypt* containers but were not able to access them. They suspect that the necessary passwords might be stored in *Bitwarden*.

Goal: Gain cleartext access to the *Bitwarden* vault to extract *VeraCrypt* passwords, if available.

5.2 Experimental Setup

As described in 3.2.4, the artefact is evaluated in an ex-post, artificial experimental setting. For each scenario, five steps are performed: (1) A dedicated virtual machine running *Windows 11* and the respective password manager application is set up; (2) The password manager vault is propagated with data corresponding to the described scenario, and the machine is put into a state as per the scenario definition; (3) To simulate the forensic data acquisition, a memory dump of the virtual machine is created (where applicable); (4) The file system of the (now turned off) virtual machine is mounted as read-only on another virtual machine (running Kali Linux) used for investigation to simulate mounting a disk image; (5) The disk image and/or memory dump are analysed using the artefact (*PMF Application Identifier* and *PMF Extractor*) to search for and extract relevant artefacts.

The results of these experiments are presented in the next chapter. Details about the software used can be found in appendix C.

5.3 Results

5.3.1 Scenario 1

First, *PMF Application Identifier* was used to scan the disk image for password manager files. The script successfully identified the *Bitwarden* desktop app and copied the *data.json* file to the extraction folder on the investigator's device.

Then, *PMF Extractor* was used to analyse both the extracted *data.json* file and the memory dump. The script successfully read relevant account information and the encrypted vault data from the JSON file. Furthermore, the script automatically extracted a possible master password from the memory dump, calculated the master key and the vault encryption key from it, and validated that the encryption key is correct. The vault data was then decrypted using the recovered key.

All results were automatically processed by *PMF Reporter*, and a summary report was created. The report gave investigators access to both passwords that have not been accessible through the GUI during seizure. Furthermore, the report contains a complete copy of all vault data as well as the suspect's chosen password generator settings.

```
# ######
# Vault Data
# ######

Facebook:
- User: suspect@gmail.com
- Password: 39EWZ!j^sgBfi9^3
- Notes:

Gmail:
- User: suspect@gmail.com
- Password: HrJ2iM$iVnyy#B&b
- Notes:
```

Figure 14: Extract from the report for scenario 1

In case the *data.json* file would not have been available, *PMF Extractor* found 30 strings that match criteria for possible *Bitwarden* vault passwords in the memory dump. Every password in the targeted vault was (among a few false positives) part of this list. Therefore, even without access to the encrypted vault data, investigators could have gained access to the desired accounts by trying a limited number of possible passwords that have been derived from memory alone.

5.3.2 Scenario 2

First, *PMF Application Identifier* was used to scan the disk image to identify *KeePass* files. The script successfully found and copied a *KeePass* database file (*Database.kdbx*), a *KeePass* key file (*Database.keyx*), and a *KeePass* configuration file (*KeePass.config.xml*).

Then, *PMF Extractor* was used to analyse the database file and the configuration file. The resulting report stated that the extracted key file belongs to the database file, together with a

master password. As no memory dump is available, examiners digitized the passwords from the notebook and run *PMF Extractor* in brute-force mode with the password list. The script automatically tried all possible passwords and found a match. Together with the key file, all vault items could be decrypted and exported to the report. The vault contained indeed an item with custom fields for the public and private keys of a *Bitcoin* wallet. Investigators were therefore able to seize these cryptocurrency assets.

Figure 15: Extract from the report for scenario 2

5.3.3 Scenario 3

PMF Application Identifier successfully identified and copied the *data.json* file from the disk image to the examiners machine. *PMF Extractor* was then used to analyse the extracted file as well as the memory dump. As the *Bitwarden* application was not running when the memory image was created, no useful artefacts could be extracted from the memory dump. However, the analysis of *data.json* revealed that the suspect had chosen to lock his *Bitwarden* vault with a PIN code, and that he had disabled the option "Lock with master password on restart".

As stated in the *PMF* pre-definition, this means that the master key is stored in the *data.json* file, encrypted with a key derived from the PIN. Examiners ran *PMF Extractor* in PIN brute-forcing mode, providing a password list with four-to-eight-digit numeric codes. Due to the small number of possible codes, a match was found within 15 minutes.

PMF Extractor automatically decrypted the master key and vault decryption key and exported all vault items. All data was made available to the examiners in the report. Among the extracted artefacts, there was indeed a password for the suspect VeraCrypt containers. This allowed examiners access to the encrypted files.

Figure 16: Extract from the report for scenario 3

5.3.4 Summary of Results

The goal of the described proof-of-concept was to evaluate the artefact for its suitability to be applied in a real-world investigation. By defining three possible investigative scenarios, and by applying the artefact to these scenarios, it has been demonstrated that *PMF* is able to extract forensically relevant artefacts related to password managers that would otherwise not be readily available to examiners.

Firstly, *PMF* can automatically scan a file system for relevant password manager files, identify these files, and extract them into a directory on the examiners machine for further analysis.

Secondly, *PMF* can extract a master password from a memory dump and use this password to decrypt and extract vault data from a locally stored data file. The artefact is therefore capable to circumvent a vault lock.

Thirdly, *PMF* can brute-force a master password or a PIN in case such information could not be extracted from a memory dump. In case of master passwords, this requires a list of possible passwords found during the investigation. In case of a weak numeric PIN code, brute-forcing is always possible.

Lastly, *PMF* can parse config files to extract the combination of key elements necessary to decrypt a vault, as well as default password generator settings. The artefact is therefore capable to provide investigators with additional potentially useful information.

Table 5: Overview of extracted forensic artefacts

Scenario	Password Manager	Extracted Forensic Artefacts
1	Bitwarden	– data.json file – Master password from memory dump – Decrypted vault data – Additional configuration data
2	KeePass	– KeePass database file – KeePass key file – KeePass config file – Master password through brute-force (from list with potential passwords secured at seizure) – Decrypted vault data (including custom fields used to store the Bitcoin public and private key) – Additional configuration data
3	Bitwarden	– data.json file – PIN code through brute-force – Decrypted master key – Decrypted vault data – Additional configuration data

6 Discussion

6.1 Summary

The aim of this thesis was to develop a process on how to extract forensically relevant data from password manager applications. Using a design science approach, an artefact was created to fulfil this purpose.

In a first step, the most common password manager applications have been identified, and *Bitwarden* as well as *KeePass* have been chosen to be in the focus of this thesis. Next, forensically relevant artefacts of these two applications have been identified by studying openly available documentation and source code. Furthermore, the applications' behaviour has been examined in an experimental setting. From the gained insights, a detailed documentation of all identified artefacts has been created, including relevant file names and paths, data structures, and encryption mechanisms.

From this documentation, the artefact *Password Manager Forensics (PMF)* was developed. Based on the existing *Vision* framework, a structured four-step approach to extract forensically relevant data from password managers was created. For each step, *Python* scripts that automatically search for, validate, and extract the defined artefacts were written.

Finally, the developed artefact was evaluated against three scenarios in an artificial, ex-post experiment. It has been shown that *PMF* is suitable to efficiently extract forensically relevant information from password managers that are, otherwise, not readily available to investigators.

6.2 Interpretation of Results

Digital forensics rely on a constantly updated and extended pool of knowledge about how to extract relevant artefacts from a variety of different applications and versions thereof. This thesis added to the active research in this domain by providing knowledge about password managers – a category of applications that has so far been neglected in digital forensics.

Although access to a suspect's login credentials could be of significant value to an investigator, no comprehensive process has yet been described on how to extract such data from seized devices. This thesis filled this gap by introducing *Password Manager Forensics (PMF)*.

PMF is based on the *Vision* framework proposed by Bang et al. [4]. In the last years, we have seen several approaches to create comprehensive processes or frameworks that combine scattered results from different research so that they can be applied in an investigation. *Vision* introduced a modular

and flexible four-step approach to extract forensically relevant artefacts from password-protected resources. Thus, it laid the ideal foundation to build *PMF* upon. This thesis took the four-step framework proposed by Bang et al. and created the necessary modules to extract forensically relevant information from password manager applications. With the presented artefact, forensic examiners have now, for the first time, a comprehensive tool available to extract such information.

PMF can efficiently identify and extract relevant files, extract a master password as well as vault passwords from a memory dump, parse configuration files for relevant data, brute-force master passwords and PIN codes, decrypt, extract, and validate password manager vault data, and create summary reports.

The result also demonstrated that password managers are a valuable source of relevant data in a forensic investigation. Despite the applications' strong focus on security, vault data can still be accessed by investigators in many situations. Namely if a device can be seized with a running vault or if a suspect has been using weak convenience features (like a PIN code), relevant artefacts can almost always be extracted. A timely capture of a memory dump can further reveal secret information even though password manager vendors are implementing measures to avoid cleartext passwords in memory as much as possible.

Due to the significance of password manager data in digital forensics, more research should be performed in this field. The presented artefact therefore comes with extendibility in mind and carries on with the flexibility of *Vision: PMF* is not seen as a final solution, but as a first part of a modular artefact that can be improved and extended to support more password manager applications and versions. To make the artefact as accessible as possible, the created knowledge has been made explicit through detailed documentation and through publishing the source-code of *PMF* to *GitHub*.

6.3 Limitations

The presented results come with several limitations. Firstly, the current version of the artefact can only be applied to *Windows* desktop applications of *Bitwarden* and *KeePass*. This makes it not directly applicable in scenarios where other applications and/or operating systems are used. While the artefact is built modularly, and the presented knowledge can be partly applied to new extensions, there is still some effort needed to expand *PMF* to include new configurations.

Secondly, the presented artefact uses unstructured memory analysis (see 2.2.1) based on *Yara* rules to search for master passwords in a memory dump. While this approach provided largely reliable results for *Bitwarden*, it was not always reliable for *KeePass*. To deliver more reliable results, the application of more complex memory analysis methods would be necessary. Due to

the highly specialized nature of such advanced memory forensic techniques, this was out of the scope of this thesis.

Thirdly, the proof-of-concept has been performed in an artificial experimental setting with a single virtualisation engine and a single operating system version. While this was the most appropriate choice for the current thesis (as discussed in 3.2.4), such settings may come with limitations regarding the generalizability and reproducibility of the results.

Lastly, the creation of the artefact was based on a qualitative data analysis by a single author. While care was taken to present all decisions as transparently and objectively as possible, a certain bias cannot be ruled out. Other researchers might have chosen different or additional forensic artefacts to extract. This possible impact on objectivity should be considered when interpreting the presented results.

6.4 Future Research

Password Manager Forensics (PMF) can be further improved and expanded in future research on three different levels. Firstly, new, or better functions for the current version of *PMF* can be created. This includes advanced memory forensic techniques that allow for a more reliable extraction of master passwords. For example, *KeeThief*, an existing *KeePass* memory extraction module [27], could be adapted for *PMF*. Another aspect that can be improved in the current version of *PMF* is the brute-force mode. The current *Python* function is far from ideal from an efficiency point of view. An improved function could leverage processor multi-threading and GPU processing, e.g., by including *Hashcat*.

Secondly, *PMF* can be expanded so that it supports other password manager applications and application versions (e.g., browser extensions, other operating systems, etc.). As discussed in 4.2, there are many options yet to pursue. From a forensic investigator's perspective, the more possible scenarios that are covered by *PMF*, the better. To add more applications to *PMF*, steps 2 and 3, as described in 3.2.3, can be repeated in future research.

Thirdly, the process in itself can be in the focus of research. *PMF* could be adapted to a future improved version of *Vision*, or a better-suited new framework. Digital forensics are constantly adapting to new technological challenges like cloud and edge computing, which make the sequential approach of first collecting data, and then analysing data, obsolete. Future research could port results from these emerging fields into the area of password manager forensics.

References

[1] A. Årnes, “Introduction,” in *Digital Forensics*, A. Årnes, Ed., Hoboken, NJ: John Wiley & Sons Ltd, 2018, pp. 1–12.

[2] A. O. Flaglien, “The Digital Forensics Process,” in *Digital Forensics*, A. Årnes, Ed., Hoboken, NJ: John Wiley & Sons Ltd, 2018, pp. 13–49.

[3] A. Case and G. G. Richard, “Memory forensics: The path forward,” *Digital Investigation*, vol. 20, pp. 23–33, Mar. 2017, doi: 10.1016/j.diin.2016.12.004.

[4] J. Bang, J. Park, and S. Lee, “Vision: An empirical framework for examiners to accessing password-protected resources for on-the-scene digital investigations,” *Forensic Science International: Digital Investigation*, vol. 40, p. 301376, Mar. 2022, doi: 10.1016/j.fsidi.2022.301376.

[5] C. Copeland, “An Integrated Framework for the Recovery of Evidence from Encrypted Hard Disk Drives,” Dakota State University, 2016. [Online]. Available: https://scholar.dsu.edu/theses/297

[6] A. Kazim, F. Almaeeni, S. A. Ali, F. Iqbal, and K. Al-Hussaeni, “Memory Forensics: Recovering Chat Messages and Encryption Master Key,” in *2019 10th International Conference on Information and Communication Systems (ICICS)*, Irbid, Jordan: IEEE, Jun. 2019, pp. 58–64. doi: 10.1109/IACS.2019.8809179.

[7] A. Fukami, R. Stoykova, and Z. Geradts, “A new model for forensic data extraction from encrypted mobile devices,” *Forensic Science International: Digital Investigation*, vol. 38, p. 301169, Sep. 2021, doi: 10.1016/j.fsidi.2021.301169.

[8] T. Groß, M. Busch, and T. Müller, “One key to rule them all: Recovering the master key from RAM to break Android’s file-based encryption,” *Forensic Science International: Digital Investigation*, vol. 36, p. 301113, Apr. 2021, doi: 10.1016/j.fsidi.2021.301113.

[9] S. Nusbaum and C. Perez, “LastPass Security Vulnerability: How Credentials are Accessed in Memory,” *TrustedSec*, Oct. 25, 2022. https://www.trustedsec.com/blog/lastpass-in-memory-exposure/ (accessed Dec. 22, 2022).

[10] Paradox Infosec, “KeePass Master Password Extraction (Version 2.52),” Oct. 31, 2022. Accessed: Dec. 22, 2022. [Online]. Available: https://youtu.be/I3-ixb9wnWg

[11] E. Casey, *Handbook of digital forensics and investigation*. Amsterdam, Boston: Academic, 2010.

[12] M. Reith, C. Carr, and G. Gunsch, “An Examination of Digital Forensic Models,” vol. 1, May 2003.

[13] N. L. Petroni, Aa. Walters, T. Fraser, and W. A. Arbaugh, “FATKit: A framework for the extraction and analysis of digital forensic data from volatile system memory,” *Digital Investigation*, vol. 3, no. 4, pp. 197–210, Dec. 2006, doi: 10.1016/j.diin.2006.10.001.

[14] A. R. Javed, W. Ahmed, M. Alazab, Z. Jalil, K. Kifayat, and T. R. Gadekallu, “A Comprehensive Survey on Computer Forensics: State-of-the-Art, Tools, Techniques, Challenges, and Future Directions,” *IEEE Access*, vol. 10, pp. 11065–11089, 2022, doi: 10.1109/ACCESS.2022.3142508.

[15] R. Shree, A. Kant Shukla, R. Prakash Pandey, V. Shukla, and D. Bajpai, “Memory forensic: Acquisition and analysis mechanism for operating systems,” *Materials Today: Proceedings*, vol. 51, pp. 254–260, 2022, doi: 10.1016/j.matpr.2021.05.270.

[16] M. H. Ligh, A. Case, J. Levy, and A. Walters, *The art of memory forensics: detecting malware and threats in Windows, Linux, and Mac memory*. Indianapolis, IN: Wiley, 2014.

[17] P. Fernández-Álvarez and R. J. Rodríguez, "Extraction and analysis of retrievable memory artifacts from Windows Telegram Desktop application," *Forensic Science International: Digital Investigation*, vol. 40, p. 301342, Apr. 2022, doi: 10.1016/j.fsidi.2022.301342.

[18] N. Mistry and M. S. Dahiya, "VolNet: a framework for analysing network-based artefacts from volatile memory," *IJESDF*, vol. 9, no. 2, p. 101, 2017, doi: 10.1504/IJESDF.2017.083978.

[19] D. K. Davis, M. M. Chowdhury, and N. Rifat, "Password Security: What Are We Doing Wrong?," in *2022 IEEE International Conference on Electro Information Technology (eIT)*, Mankato, MN, USA: IEEE, May 2022, pp. 562–567. doi: 10.1109/eIT53891.2022.9814059.

[20] L. H. Newman, "Get a Password Manager. No More Excuses," *Wired*, Jan. 03, 2018. [Online]. Available: https://www.wired.com/story/password-manager-autofill-ad-tech-privacy/

[21] Bitwarden, "World Password Day Global Survey Full Report." https://bitwarden.com/resources/world-password-day-global-survey-full-report/ (accessed Jan. 24, 2023).

[22] K. Toubba, "Notice of Recent Security Incident," *The LastPass Blog*, Dec. 22, 2022. https://blog.lastpass.com/2022/12/notice-of-recent-security-incident/ (accessed Jan. 25, 2023).

[23] C. Colby, R. Hodge, and A. Tomaschek, "Best Password Managers for 2023 and How to Use Them," *CNET*. https://www.cnet.com/tech/services-and-software/best-password-manager/ (accessed Jan. 24, 2023).

[24] T. Matthews-El, "8 Best Password Managers (January 2023)," *Forbes Advisor*, Jan. 2023, Accessed: Jan. 25, 2023. [Online]. Available: https://www.forbes.com/advisor/business/software/best-password-managers/

[25] M. Carr and S. F. Shahandashti, "Revisiting Security Vulnerabilities in Commercial Password Managers," Mar. 2020, [Online]. Available: http://arxiv.org/abs/2003.01985

[26] P. Sabev and M. Petrov, "Android Password Managers and Vault Applications: An Investigation on Data Remanence in Main Memory," 2021, [Online]. Available: https://ceur-ws.org/Vol-2933/paper31.pdf

[27] L. Christensen and W. Schroeder, "KeeThief." Accessed: Jan. 25, 2023. [Online]. Available: https://github.com/GhostPack/KeeThief

[28] P. Johannesson and E. Perjons, *An introduction to design science*. Cham: Springer International Publishing, 2014.

[29] M. Denscombe, *The good research guide: for small-scale social research projects*, 5. ed. Maidenhead: Open University Press, 2014.

[30] S. Gilbertson, "The Best Password Managers to Secure Your Digital Life," *Wired*. Accessed: Feb. 08, 2023. [Online]. Available: https://www.wired.com/story/best-password-managers/

[31] Bitwarden, "Bitwarden Security Whitepaper," *Bitwarden*. https://bitwarden.com/help/bitwarden-security-white-paper/ (accessed Feb. 13, 2023).

[32] Bitwarden, "Bitwarden Documentation," *Bitwarden*. https://bitwarden.com/help/ (accessed Feb. 14, 2023).

[33] D. Reichl, “Master Key,” *KeePass*. https://keepass.info/help/base/keys.html (accessed Feb. 28, 2023).

[34] J. van der Velde, “Keepass decryption,” *GitHub*, Feb. 01, 2023. https://github.com/scubajorgen/KeepassDecrypt (accessed Mar. 01, 2023).

[35] D. Reichl, “Security,” *KeePass*. https://keepass.info/help/base/security.html (accessed Mar. 01, 2023).

[36] M. Cone, “Basic Syntax,” *Markdown Guide*. https://www.markdownguide.org/basic-syntax/ (accessed Feb. 16, 2023).

[37] VirusTotal, “Writing YARA rules.” https://yara.readthedocs.io/en/stable/writingrules.html (accessed Feb. 23, 2023).

Appendix A: Reflection Document

How does your study correspond to the goals of the thesis course? Why? Focus on the goals that were achieved especially well and those that were not well achieved.

This thesis corresponds to the course goals in terms of that I could independently complete an academic paper that contributes to knowledge in computer and system sciences. This thesis builds on scientific literature and summarizes and extends an existing framework (*Vision*). It fills a knowledge gap by addressing a category of software that has not yet been in the focus of forensic research. This was done by creating an artefact through applying the design science framework. The artefact was shown to be suitable to fulfil the research aim within the defined limitations. These limitations in turn suggest possible directions for future research, which are discussed at the end of the thesis.

The course goals also mention reflecting and discussing ethical aspects of the work. While these aspects are addressed in my thesis, they are not a central part. The creation of a software artefact and the evaluation in an artificial setting did not involve human participants nor strong ethical dilemmas. Those goals were, therefore, not addressed as thoroughly as the remaining ones.

How did the planning of your study work? What could you have done better?

The planning of my work started with a short concept paper including a summary of my research idea, a possible rough structure of the thesis, as well as a time schedule. This plan was discussed with and approved by my supervisor. Based on this plan, I was able to follow the different steps of the thesis and complete them on time. Having worked as a project manager for several years, I am aware of the importance of having a solid plan before starting a project and taking the time for regular controlling and adjustments.

How does the thesis work relate to your education? Which courses and areas have been most relevant for your thesis work?

I chose to author my thesis in the field of digital forensics as the two courses addressing this field (Digital Forensics, Cyber Forensics) were among the ones that sparked my interest the most. However, thoroughly analysing password managers – software with a high focus on security – allowed me to apply concepts and practical knowledge I gained during the whole programme in a comprehensive and practical way. Furthermore, I chose the design science framework to create my artefact. This allowed me to apply the knowledge gained from the respective methodology course in a larger project.

How valuable is the thesis for your future work and/or studies?

The present thesis allowed me to set up and implement a practical project in the realm of information security and to document and discuss it in depth. Writing this thesis combined aspects of project management, application of academic knowledge in a practical context, as well as communication and critical thinking. Therefore, I believe that the experience of writing this thesis is valuable for my future work. Furthermore, the use of the design science framework resulted in the creation of a practical artefact. By publishing this artefact to *GitHub*, it is not only available to other scientists but also a valuable addition to my portfolio.

How satisfied are you with your thesis work and its results? Why?

In the present thesis, I successfully identified a knowledge gap, explicated a research problem from it, and created a working artefact to address this problem. I am, therefore, very satisfied with the result of my thesis. Having had only basic knowledge of *Python*, and almost no knowledge about the inner workings of password managers, authoring this thesis gave me a great opportunity to make a deep dive into unknown waters. The writing process showed me that I was able to take academic knowledge, use it to address a practical problem, and successfully create a working solution to said problem while simultaneously improve my knowledge in various areas. Finally, bringing all of this together in the thesis paper to present research in a so far not researched area has been highly rewarding.

Appendix B: List of Password Managers

	Operating Systems (Stand-Alone App)						Browser Plugins				Other Apps	
Name	**Windows**	**macOS**	**Linux**	**Android**	**iOS**	**ChromeOS**	**Chrome**	**Firefox**	**Safari**	**Edge**	**Web-App**	**CLI**
1Password	x	x	x	x	x	x	x	x	x	x		x
Bitwarden	x	x	x	x	x		x	x	x	x	x	x
Dashlane				x	x		x	x	x	x	x	x
Enpass	x	x	x	x	x		x	x	x	x		
KeePass(XC)[17]	x	x	x	x	x		x	x		x	x	x
Keeper	x	x	x	x	x		x	x	x	x	x	x
LastPass	x	x		x	x		x	x	x	x	x	x
pass[18]	(x)	(x)	x	(x)	(x)		(x)	(x)				x
NordPass	x	x	x	x	x		x	x	x	x	x	
Norton PM				x	x		x	x	x	x		
Roboform	x	x		x	x		x	x	x	x		

					Recommended		
Name	**Open-Source**	**Security Whitepaper**	**Type**	**Website**	**Forbes**	**Cnet**	**Wired**
1Password	Partially	Yes	Cloud	https://1password.com	x	x	x
Bitwarden	Yes	Yes	Cloud	https://bitwarden.com/	x	x	x
Dashlane	Partially	Yes	Cloud	https://www.dashlane.com/	x	x	x
Enpass	No	Yes	Local	https://www.enpass.io			x
KeePass(XC)	Yes	Yes	Local	https://keepass.info	x	x	x
Keeper	No	Yes	Cloud	https://www.keepersecurity.com	x	x	x
LastPass	No	Yes	Cloud	https://www.lastpass.com	x		x
pass	Yes	No	Local	https://www.passwordstore.org/			x
NordPass	No	Yes	Cloud	https://nordpass.com	x		x
Norton PM	No	No	Cloud	https://my.norton.com/extspa/passwordmanager	x		
Roboform	No	Yes	Cloud/Local	https://www.roboform.com			x

[17] *KeePass* is an open-source project actively developed by a large community. For this overview, the "*KeePass* ecosystem" has been treated like a single application, therefore including community-developed parts such as mobile apps (e.g., *KeePass2Android*, *Strongbox*), or browser extensions.
[18] *pass* is an open-source CLI password manager that organises credentials in GPG encrypted files. The pass file structure is used by several community-developed interfaces that are compatible with *pass* but, unlike *KeePass*, are not forked versions. These interfaces are marked here as (x).

Appendix C: Used Software

Host System

Microsoft Windows 11 Pro, 64-bit, Version 22H2
Oracle VirtualBox 7.0.6 / VirtualBox Guest Additions 7.0.6

Virtual Machine (Scenarios)

Microsoft Windows 11 Education, 64-bit, Version 22H2
VM-Setup: 2 CPU, 4096 MB Memory

Virtual Machine (Examination)

Kali Linux 2022.4 (Kernel Linux 6.1.0-kali5-amd64)
VM-Setup: 4 CPU, 8192 MB Memory

Password Managers

- Bitwarden Windows Desktop App (Standard Installer), Version 2023.1.1[19]
- KeePass, Installer for Windows, Version 2.53.1[20]

Other Software

Memory dumps have been created by VirtualBox's own VBoxManager[21].
Scripts were run using Python 3.11.2

Software used during the creation of the artefact:
- PyCharm Community Edition 2022.3.2 (Python scripts)
- HxD Hex Editor 2.5.0.0 (analysis of memory dumps)
- Atom 1.60.0 (analysis of json files)

[19] Downloaded from https://bitwarden.com/download/
SHA-256: a2601edc4230a8dde601e6d14f5d1d98264f96d84609de913d318a557ed60ee1
[20] Downloaded from https://keepass.info/download.html
SHA-256: 067727caa782f53f6232f8f59bc945384fce98817b014300039b28487c06a5cd
[21] Command: VBoxManage.exe debugvm "Thesis_VM" dumpvmcore --filename memorydump

Appendix D: Artefact File Structure

The following list gives an overview about the file structure of the artefact. The full source code of *Password Manager Forensics (PMF)* is available on GitHub:
https://github.com/shaehni/password-manager-forensics

Password Manager Forensics:

pmf_appident.py	PMF Application Identifier
pmf_extractor.py	PMF Extractor
requirements.txt	Python dependencies
—Definitions*	
bitwarden_functions.md	Documentation for bitwarden.py
bitwarden_patterns.md	Pattern definition for Bitwarden
keepass_functions.md	Documentation for keepass.py
keepass_patterns.md	Pattern definition for KeePass
pmf_apps.json	Relevant file paths and names for all applications
pmf_scripts.md	Documentation for pmf_appident.py, pmf_extractor.py, pmf_reporter.py
—Modules	
bitwarden.py	Functions to find, analyse and export Bitwarden artefacts
bitwarden.yara	Yara definition for Bitwarden memory artefacts
keepass.py	Functions to find, analyse and export Keepass artefacts
keepass.yara	Yara definition for KeePass memory artefacts
pmf_reporter.py	PMF Reporter module

* All definitions can be found in the following chapter (appendix E).

Appendix E: Pre-Definition of Forensic Password Manager Artefacts

01 – pmf_apps.json

```
{
  "Bitwarden": {
    "Desktop App": {
      "absolute_paths": {
        "directories": [
          "Users/*/AppData/Roaming/Bitwarden"
        ],
        "files": {
          "Bitwarden Data File": "data.json"
        }
      }
    },
    "Store App": {
      "absolute_paths": {
        "directories": [

"Users/*/AppData/Local/Packages/8bitSolutionsLLC.bitwardendesktop_*/LocalCache/Roam
ing/Bitwarden"
        ],
        "files": {
          "Bitwarden Data File": "data.json"
        }
      }
    },
    "Portable App": {
      "relative_paths": {
        "directories": [
          "bitwarden-appdata"
        ],
        "files": {
          "Bitwarden Data File": "data.json"
        }
      }
    }
  },
  "KeePass": {
    "Desktop App": {
      "absolute_paths": {
        "directories": [
          "Users/*/AppData/Roaming/KeePass"
        ],
        "files": {
          "KeePass Configuration": "KeePass.config.xml"
        }
      },
      "relative_paths": {
        "directories": [
          ""
        ],
        "files": {
          "KeePass Database File": "*.kdbx",
          "KeePass Key File": "*.keyx"
        }
      }
    }
  }
}
```

02 – pmf_scripts.md

Password Manager Forensics – Scripts

PMF includes different scripts to automate the extraction process of forensic artefacts from password manager applications. This file documents these scripts.

Application Identifier

`pmf_appident.py`: This script loads known application paths and relevant filenames from a pre-definition file and searches the file system for them. Identified relevant files are copied into a folder. Arguments:

- `--search-path [file path]`: Change the root directory to search for artefacts (default: C:/)
- `--extract-to [file path]`: Define the folder where artefacts are extracted to (default: ./extract)
- `--predefinition [file path]`: Path to the pre-definition file (json) (default: Definitions/pmf_apps.json)

Extractor

`pmf_extractor.py`: This script scans the extracted files and/or a memory dump for forensic artefacts, analyses and processes the artefacts, and creates a report with the results. Arguments:

- `--report [folder]`: Path to the directory where the report will be saved (default: ./report)
- `[password manager]`: Select the password manager application from which artefacts shall be extracted (Available in this version: bitwarden, keepass). Every password manager module accepts application-specific arguments. See the respective module description for further information.

Reporter

`pmf_reporter.py`: This module defines the `Report` class and corresponding functions. A `Report` is instantiated by providing a list of chapters. Functions:

- `add(chapter, element)`: This function adds data that will be printed under the given chapter. The element argument is a dictionary `{name: value}` where `name` is the string how it will appear in the report, and `value` is either a string, a list, or a dictionary.
- `save(path, name)`: This function creates a report file containing all data elements stored in the `Report` object. The file is created under the given `path`. The current date and time are automatically added to the `name` of the file.

The created report file is a text document containing all chapters as headings with corresponding data listed beneath. Chapters are sorted according to the list order when the object was instantiated. Chapter data is sorted in the order it was added.

03 – bitwarden_patterns.md

Bitwarden Search Pattern Definition

This document defines the forensically relevant artefacts, and their structure, which can be found in files from the Bitwarden application directory and in a memory dump.

data.json

This file stores configuration information as well as encrypted keys and vault data in JSON format. It is stored in `[application directory]/data.json`.

Relevant Artefacts

- `['global']`
  - `['rememberedEmail']`: This element stores the last email address that was used to log into Bitwarden if the corresponding box was ticked.
- `['authenticatedAccounts']`: This element contains an array of all accounts that have been authenticated. Account IDs match the pattern `[a-z0-9]{8}-([a-z0-9]{4}-){3}[a-z0-9]{12}`.
- `['(accountID)']`
  - `['data']`
    - `['ciphers']['encrypted']`: This element contains a list of all vault items in encrypted format. Vault item IDs follow the same pattern as Account IDs (`[a-z0-9]{8}-([a-z0-9]{4}-){3}[a-z0-9]{12}`). See the chapter below for more details about encrypted vault data.
    - `['passwordGenerationHistory']['encrypted']`: This element contains an array of recently generated passwords (in encrypted format) as well as the time of creation. This information is not available through the GUI.
  - `['keys']`: This element contains a list of encrypted keys, namely the `cryptoSymmetricKey` and the `privateKey`. See the chapter below for more details about the key encryption format.
  - `['profile']`: This element includes a list with information about the user account, most notably the full name (`['name']`) and the account email (`['email']`). Furthermore, the number of `['kdfIterations']` needed to calculate the master key as well as the last time of vault synchronisation `['lastSync']` is available.
  - `['protectedPin']`: If a user has set a PIN with the option "Lock with master password on restart", this element contains the PIN code in the encrypted key format (see below).
  - `['pinProtected']['encrypted']`: If a user has set a PIN without the option "Lock with master password on restart", this element contains the master key encrypted with an encryption key derived from the PIN. This brings a potential for brute-forcing the master key if a weak PIN is set (see below).
  - `['passwordGenerationOptions']`: This element contains the settings last used for the password generator. This could be valuable information in case that vault data cannot be retrieved and brute-forcing of account passwords is necessary. If special characters are selected, only the following are used: `!@#$%^&*`

Availability

The list of authenticated accounts as well as all data in the `['(accountID)']` element is available when the app is closed, or the vault is locked. However, it gets purged when a user signs out. The

remembered email stays available even after sign-out. The list of generated passwords is not stored on the server. It is permanently lost when a user signs out.

Encrypted Vault Data

Bitwarden stores encrypted data as an what they call encrypted string. These encrypted strings have the following format: `(Encryption Type [int]).(Initialization Vector [Base64])|(Encrypted Data [Base64])|(Message Authentication Code [Base64])`.

In all observed data, the encryption type was always `2`, which corresponds to `AesCbc256_HmacSha256_B64`.

Encrypted vault data can be matched by the following regex expression: `2\.[a-zA-Z0-9+\/]{22}={0,2}\|[a-zA-Z0-9+\/]+={0,2}\|[a-zA-Z0-9+\/]{43}={0,2}`

Encrypted Keys

Keys are encrypted the same way as encrypted vault data. While vault data is encrypted with the symmetric key, the stored symmetric key is encrypted with the expanded master key. The encryption key is generated by expanding the master key through HKDF with a salt of `enc`. The MAC is generated by expanding the master key through HKDF with a salt of `mac`.

The first 32 bytes of the symmetric vault key contain the encryption key. The second 32 bytes contain the MAC.

Encrypted PIN

If the user has set a PIN and has disabled the option "Lock with master password on restart", the master key is encrypted with a key derived from the PIN and stored in the `data.json` file. This encryption key is calculated identically as the original (stretched) master key. In case a weak PIN is used, the master key can be brute-forced and the vault can be decrypted without knowledge of the master password.

Memory

Master Password

The master password appears in memory as part of the following string: `com.bitwarden.vault [account email address] [master password] -`

Vault Passwords

Vault credentials (passwords) are stored in memory in a data structure with the format `CD 25 00 00 03 00 00 00 [XX] 00 00 00 [Password] [Random Bytes] CD 25` where `[XX]` indicates the length of the following string (possible password). Searching for this structure results in false positives. These are reduced by only considering strings that match the characters used by Bitwarden's password generator (`A-Za-z0-9!@#$%^&*`), and filtering out unlikely strings (<8 characters, only numbers, or only capital letters). Furthermore, strings that contain certain substrings that have been observed in false positives are filtered out.

04 – bitwarden_functions.md

Bitwarden Module Definition

This document describes the scripts and functions available to extract relevant forensic artefacts from the Bitwarden Windows Desktop application. All functions are defined in the file `Modules/bitwarden.py`.

Arguments for Extractor

PMF Extractor accepts the following Bitwarden-specific options:

- `--data-file`: Location of the `data.json` file.
- `--memory-dump`: Location of the memory dump file.
- `--master-pw`: If known, the master password can be manually provided to decrypt vault data.
- `--brute-force`: Text file with one password per line for brute-forcing the master password.
- `--brute-force-pin`: If the user used a PIN to unlock the vault and disabled the option "Lock with master password on restart", the PIN can be brute-forced instead of the master password.

Discovery

- `get_file_artefacts(data_file, Report)`: Loads the given `data.json` file, extracts the relevant forensic artefacts as defined in the pattern pre-definition file, and adds results to the report object.
- `scan_memory_for_pw(memory_dump_file)`: Loads the given memory dump file and uses Yara rules in `bitwarden.yara` to scan the file for artefacts as defined in the pattern pre-definition file. Returns a string with the master password, and a list of likely vault passwords.
- `get_pbkdf_iterations(email)`: If the number of PBKDF iterations could not be found in `data.json`, this fallback function will query the Bitwarden login API for the PBKDF setting for a given account email address. Returns an integer or False.

Analysis

- `Ciphertext`: This class allows loading an encrypted vault data item from a string as it is stored in `data.json`. On instantiation, the string is split and base64 encoded fields are decoded. A Ciphertext object consists of four variables: `encryption_type`, `iv`, `data`, `mac`.
- `Decryptor`: This class can be instantiated by providing the encryption key part and the mac part of the symmetric vault key. Once instantiated, Ciphertext elements can be decrypted using the `decrypt` function. Data is validated through HMAC.

Post-Processing

The Bitwarden module creates a report object with the chapters `Account Info`, `Cryptography`, `Vault Data`, `Password Generator`, `Memory Analysis`. All results are added to the report object during discovery and analysis. The report is created by the report objects own create function (see `pmf_scripts.md` for further information).

05 – keepass_patterns.md

KeePass Search Pattern Definition

This document defines the forensically relevant artefacts, and their structure, which can be found in KeePass files and in a memory dump.

database.kdbx

A KeePass database file contains all vault data. This file usually has the file ending `.kdbx` and the following file header: `03 D9 A2 9A 67 FB 4B B5`

Relevant Artefacts

- Header bytes 8-9: File format version of the database (minor)
- Header bytes 10-11: File format version of the database (major)
- The rest of the header contains variable-length meta data in the form: `Type` (1 byte), `Length` (2 bytes), `Data` (according to length). The following types are of forensic relevance:
  - `03`: Compression flag (gzip)
  - `04`: Master seed
  - `05`: Transform seed
  - `06`: Transform rounds
  - `07`: Initialisation vector
  - `00`: End of header
- Rest of file: Payload area (encrypted vault)

Availability

The database meta data is part of the KeePass database file. It is therefore always available in an intact file.

Encrypted Vault Data

Vault data is stored as encrypted XML. Decryption is performed by first creating a composite key from the master password and/or key files. Then creating the master key through AES transformation and by using the master seed. A decryption context is then set up using the master key and the initialisation vector. The process is described in detail here. PMF uses the external pykeepass library to perform vault decryption.

KeePass.config.xml

The KeePass config file contains settings about the KeePass application. It is saved in the application directory and has the file name `KeePass.config.xml`. As the file ending suggests, it is in XML format.

Relevant Artefacts

- `<Configuration><Defaults><KeySources><Association>`: This element stores the path to the last opened database files (`<DatabasePath>`), whether a master password is set for the respective file (`<Password>`), and the corresponding key file, if applicable (`<KeyFilePath>`). The setting to store this information can be turned off by a user, but it is enabled by default.

- `<Configuration><Application><MostRecentlyUsed>`: This element stores the most recently used database files as `<Items><ConnectionInfo><Path>`.
- `<Configuration><PasswordGenerator>`: This element stores the last used settings of the password generator.

Availability

The configuration file is stored in the application directory as cleartext XML.

Memory

Master Password

In the analysed memory dump, the master password appeared as part of the following pattern: `5E DF 27 D1 00 3B 00 94 [Master Password]`, followed with at least 10 `00` bytes, with a `00` padding between every character of the password. To reduce false positives, unlikely strings (<8 characters, non-printable characters) are filtered out from the results. However, as KeePass implements measures to avoid cleartext leaks of the master password in memory, this pattern matching method is not reliable.

Vault Passwords

Vault credentials (passwords) are stored in memory in a data structure with the format `00 A0 57 ?? ?? ?? 7F 00 00 [XX] 00 00 00 00 00 00 00 [Password] 00 00` where `[XX]` indicates the length of the following string (possible password). Searching for this structure results in false positives. These are reduced by filtering out unlikely strings (<8 characters, non-printable characters, only numbers, or only capital letters).

06 – keepass_functions.md

KeePass Module Definition

This document describes the scripts and functions available to extract relevant forensic artefacts from the KeePass Windows Desktop application. All functions are defined in the file `Modules/keepass.py`.

Arguments for Extractor

PMF Extractor accepts the following Bitwarden-specific options:

- `--database`: Location of the KeePass database file (`.kdbx`).
- `--memory-dump`: Location of the memory dump file.
- `--config-file`: Location of the KeePass configuration file (`KeePass.config.xml`)
- `--master-pw`: If known, the master password can be manually provided to decrypt vault data.
- `--key-file`: Location of the key file that is part of the master key.
- `--brute-force`: Text file with one password per line for brute-forcing.

Discovery

- `get_database_info(database_file, Report)`: Reads the (binary) header data from a KeePass database file and extracts relevant information (File version, Master seed, Transform seed and rounds, Initialisation vector)
- `parse_config_file(config_file, Report)`: Parses the KeePass configuration file and extracts relevant information (last used databases and their encryption configuration).
- `scan_memory_for_pw(memory_dump_file)`: Loads the given memory dump file and uses Yara rules in `keepass.yara` to scan the file for artefacts as defined in the pattern pre-definition file. Returns a tuple containing a list of possible master passwords, and a list of likely vault passwords.

Analysis

- `pykeepass`: PMF uses the external module Python module pykeepass to decrypt the database file and to export vault items. First, a `PyKeePass` object is created by providing necessary decryption information. Then, by iterating over all vault items, vault data is extracted and sent to the reporter module.

Post-Processing

The KeePass module creates a report object with the chapters `Database`, `Cryptography`, `Vault Data`, `Password Generator`, `Memory Analysis`. All results are added to the report object during discovery and analysis. The report is created by the report objects own create function (see `pmf_scripts.md` for further information).

Appendix F: Example Report

The following is an example output from *PMF Reporter*. All data is fictious and not used in any real-world application.

```
# ######
# Database
# ######

Database File: extract/KeePass/Desktop App/Database.kdbx
File Version: 3.1

Header Information:
- Compressed (gzip): True
- Master Seed: kPDcZ87Uh0U68HSgij3iGXYQGQmicWEwfzsbBVd3PhQ=
- Transform Seed: XmEdB4WN6AIlbqUFQHeiWhOfQb8Nexd1xPXCNuPCISc=
- Transform rounds: 60000
- Initialisation vector: VmwSHkW7rwNn0k+I6Pinrg==


# ######
# Cryptography
# ######

Master Password: m@sterPa$$word-KP-584
Key File: extract/KeePass/Desktop App/Database.keyx
Master Key: k7ISY3TdC1e65JMf1oGAxokdJNwtBeqiREo/F1u8mPQ=


# ######
# Vault Data
# ######

Test-Login1:
- User: test.user@gmail.com
- Password: EPkbRT2quyIZ5iipaoga
- Notes: This is a note.
- Custom Fields: {}

Test-Login2:
- User: test.user2
- Password: DriR6AS2EWOd4X46tGTV
- Notes: This is another note.
- Custom Fields: {}

Test-Login3:
- User: user33
- Password: gx9ZUiW2Xbfa3EBjUiNr
- Notes: Another note for this login.
- Custom Fields: {}

Bitcoin Wallet:
- User: None
- Password: None
- Notes: None
```

- Custom Fields: {'Address': '16bjbPEvLXMJwg6bs79UxTZmMsJC3mh2K', 'Private Key': 'L5FU1KCRveKDZdK2SR4o76YqRUoecVQiNjYBjYanEZw8JEeH97vF'}

######
Config
######

Database 0:
- Database Path: ..\..\Users\Thesis\Documents\Database.kdbx
- Password used: true
- Key File: ..\..\Users\Thesis\Documents\Database.keyx

Database 1:
- Database Path: ..\..\Users\Thesis\Documents\Testdb.kdbx
- Password used: true

Last Used Database Files:
- ..\..\Users\Thesis\Documents\Database.kdbx
- ..\..\Users\Thesis\Documents\Testdb.kdbx

Password Generator Settings:
- Name: (Custom)
- GeneratorType: CharSet
- Length: 20

######
Memory Analysis
######

Memory Dump File: /mnt/transfer/Thesis/Proof-of-Concept/Scenario 2/mem.raw

Possible Vault Passwords:
- 0000009e3fd1
- EPkbRT2quyIZ5iipaoga
- BD56D950EF143E27ED10552C283E024AD3ADE6CD1EA3835DD7395B3D9C41C885
- primordials
- Organization
- 00000034b2a6
- 16bjbPEvLXMJwg6bs79UxTZmMsJC3mh2K
- Preferences
- FileHandleCloseReq
- G3SXF27YLA5TN5ORSGWK4NBSWEXHIBCW
- sFTWJgQvESWb4r
- 39EWZ!j^sgBfi9^3
- DriR6AS2EWOd4X46tGTV
- aGQb2w5C2QphxC
- \\WINDOWSMA
- .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
- gx9ZUiW2Xbfa3EBjUiNr
- Heartbeat

Report created on 2023-03-10 13:06 by PMF Reporter