Computer Science • 26(4) 2025 https://doi.org/10.7494/csci.2025.26.4.6822

Kamil Jurczyk
Paweł Topa
Łukasz Faber

# TOWARD RAM FORENSICS SUPPORTED BY MACHINE-LEARNING METHODS

**Abstract** *In this article, we propose an enhancement to the computer forensics technique of using Machine-Learning tools to analyze the contents of RAM in order to extract information that is potentially useful during an investigation. In the specific case presented, the use of the extracted information to generate more-optimal dictionaries for dictionary cryptanalysis is considered. Increasing user awareness is making cryptanalysis of passwords increasingly difficult for law enforcement. Long and complex passwords are impossible to crack – even when high-performance computing platforms are available. A sensible method of optimization is to look for hints to use a dictionary that contains text phrases more likely to be used in the specific case under attack. Such a hint could be an analysis of RAM taken from a suspect computer. Machine-learning methods can significantly facilitate this task. In this article, we also explore the effectiveness of such an approach and its usefulness in practical applications. We also consider applications of the proposed approach for other purposes, such as OSINT.*

**Keywords** forensic, dictionary attacks, machine learning

**Citation** Computer Science 26(4) 2025: 77–103

## 1. Introduction

As one of the sub-fields of Computer Science, computer forensics [11] is a key element used by law enforcement agencies as well as others who are involved in the detection of digital breaches or data recovery. According to current trends, the amounts of fraud and numbers of computer crimes are increasing every year [4]. Computer forensics involves the application of investigative techniques to analyze and recover potential evidence from electronic devices [20]. This process includes identifying, preserving, analyzing, and presenting digital evidence that can be used in legal proceedings. Computer forensics extends beyond just computers to encompass a wide range of devices like smartphones, cameras, IoT devices, and network devices. The significance of computer forensics lies in its ability to collect and examine data crucial for proving cybercrimes, aiding law enforcement in bringing criminals to justice.

Sometimes the investigation of violations of the law requires access to encrypted material that is, in practice, the breaking of security measures, such as a password for a device, file, or website. To facilitate this process, various types of tools are available to enable the analysis or to overcome the security necessary to continue the investigation. Depending on the type of action, specific methods, tools, and scope can be chosen. In each case, the desired outcome will be to correctly analyze the evidence gathered and prove or deny the allegation of a computer crime.

Some of the tested materials may have the aforementioned safeguards that prevent efficient analysis. Based on the collected information, one of the most common results of acquiring data from evidence is encrypted files consisting of an unknown number of characters. These files are most often the result of encrypting users of a given workstation, which contains information sensitive to the perpetrator but crucial for the entire criminal process.

Since modern ciphers such as AES are practically unbreakable by using analytical methods, the only way of getting access to encrypted data is by finding (guessing) cryptographic keys. These keys, used for encrypting files (or, more generally speaking, data containers), are produced by using functions called KDFs (*Key Derivation Functions*). The KDF functions take input passphrases and produce cryptographic keys having desired parameters (especially length) or hashes that hide original passphrases. In practice, the KDF functions use one-way digest functions (i.e., SHA-2 or SHA-3 digest functions) or more-specialized functions that have components additionally increasing their security [2, 15].

As in any war in which there is a race between methods of attack and means of defense, so too in the case of attacks on hashed passwords, the use of powerful computing platforms and highly efficient implementations is countered by algorithms with a high cost of attack. On the one hand, the defense of privacy is a human right, and on the other hand, the security of society can only be guaranteed by legitimate violations of this privacy – against suspected criminals.

The aforementioned strengthening of password security by increasing the cost of attacks can be offset by optimizing the attacks themselves. Strengthening the

computational power involved is effective to a limited extent – as a password length increases, the computational effort required increases exponentially. Optimizing the attack is relatively difficult if the suspect has followed the recommended password usage rules: use a password manager, and use long random passwords. To improve their chances of success, the investigator should use any technique that could hint at the form of the password.

In order to compete with cyber criminals, computer forensics must use every possible piece of information to gain an advantage. As noted in [9], the size of operational memory is becoming so large that it can contain valuable information from a computer forensics point of view. Among other things, the authors of this publication list processor information, open files and registry handles, files currently in use, network traffic information, passwords and cryptographic keys, decrypted data, and other data. Also, there are many ways (software and hardware) to retrieve and save a memory image. The importance of computer forensics focused on RAM analysis has also been proven by publications; i.e., [12, 16, 18].

In this article, we propose a method for gaining hints about passwords in the situation of accessing a memory image (RAM in particular) obtained from a suspect. We propose an enhancement to the computer-forensics technique of using Machine-Learning tools to analyze the contents of RAM in order to extract information potentially useful during an investigation. In the specific case presented, the use of extracted information to generate more optimal dictionaries for dictionary cryptanalysis is considered. The increasing awareness of users makes cryptanalysis of passwords increasingly difficult for law enforcement. Long and complex passwords are impossible to crack – even when high-performance computing platforms are available. A sensible method of optimization is to look for hints to use a dictionary that contains text phrases more likely to be used in the specific case under attack. Such a hint could be an analysis of RAM taken from a suspect's computer. Machine-learning methods can significantly facilitate this task. In this article, we also explore the effectiveness of such an approach and its usefulness in practical applications. We also consider applications of the proposed approach for other purposes, such as OSINT.

The organization of the article is as follows: the next section briefly discusses the issues of attacks against hashed passwords in a more detailed way. Section 4 provides an overview of forensics tools useful in analyzing memory dumps. Section 6 presents the authors' proposed scheme for memory image analysis. The article ends with concluding remarks summarizing the results and observations.

## 2. Challenges related to cryptanalysis of hashed passwords

Passwords are one of the oldest methods of controlling access to IT resources. Despite their shortcomings and the development of other methods such as biometrics, we are not abandoning this method. Its main disadvantage stems from the frailty of the human mind, which cannot cope with memorizing randomly structured data. In

order to remember a phrase that acts as a password, we choose words or groups of words that are easy to remember.

Passwords are stored in the computer system in hashed form; i.e., a so-called hash is calculated for a given text phrase. This can be the result of a cryptographic hash function (e.g., SHA-2 or SHA-3) or a dedicated KDF (Key Derivation Function) (e.g., scrypt [15]).

An investigator wants to access unclassified passwords by having their hashes. The basic and simplest method of breaking through the security of an established password is a brute-force attack that involves checking all possible characters, numbers, words, or keys in the hope of hitting the right combination that will allow access to the selected file, system, or device. Such attacks are usually performed automatically and are characterized by the long time required to achieve the required effect. Due to several factors, determining the exact time to obtain the right password varies depending on parameters such as the length of the password itself; the longer the password, the harder it is to find the right combination of characters. In addition, the special characters often found in passwords make it more difficult and expand the area that needs to be checked. Keep in mind that, depending on one's hardware resources, the time for a successful brute-force attack can vary significantly.

It is practically impossible to recover a hashed password from its hash because the hash functions and KDFs used today are one-way and we are not aware of effective methods of attacking this property. The only practically applicable method is dictionary cryptanalysis. It is a variant of a brute-force attack where, instead of testing the entire input space, only phrases that could potentially be passwords are checked. A dictionary attack therefore needs a hint in the form of a list of plaintext passwords for which the attacker will check the resulting hashes and compare with the attacked hash. Success is achieved if the hashes are the same – the result is a phrase from the dictionary. The leading programs designed for such attacks are Hashcat https://hashcat.net/hashcat/ or JohnTheRipper https://www.openwall.com/john/.

To succeed in a dictionary attack, hundreds of millions of phrases or more need to be checked, which is a computationally intensive task. Currently, the main component that allows efficient password breaking is the computer's GPU. These types of processors are designed to process in parallel huge amounts of data. Such a type of processing is required in computer graphics. Luckily, many other problems can be efficiently solved using this way of processing. Brute-force cryptoanalysis is among them. The GPUs allow us to test in parallel a huge number of potential passphrases.

## 2.1. Preventing attacks on hashed passwords

Another important issue related to dictionary attacks against KDF functions is a technique for strengthening these functions. Functions such as scrypt and Argon2 have properties called *memory hardness* – such that the function (algorithms) are called memory-hard [3]. It refers to the property of the KDF function that requires a significant amount of memory to compute, which significantly increases the cost of carrying

out a brute-force attack – even if parallel computing (in particular, GPU computing) is used. This property makes brute-force attacks unprofitable, while honest users still perform authentication in a reasonable amount of time. Bear in mind that an adversary wants to quickly test several million potential passwords, while a legitimate user only checks one password. Key derivation functions without this feature are not considered to provide a sufficient level of security against dictionary attacks [1].

The concept of memory hardness is a combination of time complexity and memory units consumed [17]. An adversary can make modifications to the function's algorithm to speed up the calculation, but only at the expense of increasing the memory complexity of the algorithm. Similarly, a reduction in the memory requirements of the algorithm can only come at the cost of an increase in computational effort.

## 2.2. Improving efficiency of dictionary attacks

Dictionary attacks are a kind of optimization of brute-force attacks. Instead of testing an entire input message space, we test only possible input data. Primarily, this means all text phrases can be used as user passwords (for authentication) or cryptographic keys (i.e., for container encryption). This significantly reduces the input message space; if the password is long enough, however, the adversary has to test a huge number of candidates. For example, a 15-character password with numbers and upper- and lower-case letters will be broken after several million years.

The effectiveness of a dictionary attack depends on the quality of the passwords used by the user. Long randomly generated passwords are almost impossible to crack; however, they are very difficult to remember. Users often use typical and easy-to-remember text phrases, e.g., '123456,' 'admin,' 'qwerty' (https://en.wikipedia.org/wiki/List_of_the_most_common_passwords). More aware users use passwords based on combinations of easy-to-remember character strings from their social environment; e.g., names of loved ones, literature, and films. The most advanced users use password managers.

Dictionaries for dictionary attacks are available on the Internet for free use. They contain passwords from various leakages that happened in the past. The `rockyou.txt` file is one of the well-known examples of a dictionary: https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt). Dictionaries are usually very large files, so software designed to perform dictionary attacks has alternative mechanisms for generating potential passwords; e.g., using rules.

Recently, artificial-intelligence methods have been applied to generate customized vocabularies based on evidence [10, 13, 14, 19]. Models based on deep learning require some data to train. After a successful learning phase, the model generates potential passwords that may be more similar to those used by the attacker. Thus, the quality of the training material is crucial for success in breaking passwords.

Success in cracking passwords can therefore be achieved by obtaining additional information from various sources to personalize the dictionary and test only the most likely phrases. The source of this data can be any analysis relating to the target

of the attack (open-source intelligence) as well as collected evidence (if the target is encrypted, evidence of a crime).

## 3. Challenges related to RAM analysis

Seemingly, the biggest challenge is getting an image of the RAM, which should be wiped out when the power is cut. Indeed, this poses a problem, although various methods of recovering data from memory after a power cut have been proposed; e.g., [7,8]. In this article, however, we skip over the issues of memory-image acquisition, focusing instead on analyzing what we manage to acquire by various means.

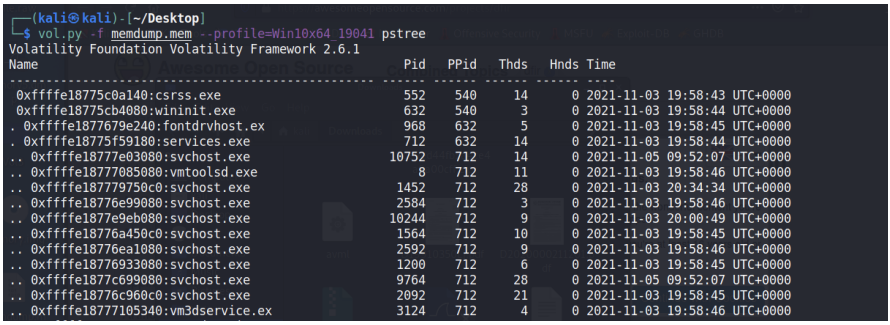
The main factor responsible for selecting the right analysis tool is the data that needs to be properly processed, handled, and presented. The vast majority of this comes from the binary copies of media created for an investigation. However, this process is insufficient and leaves a large field that can be filled in by information from a device's RAM. Currently, there are not many tools on the market that enable RAM analysis [6]. Among the most popular, we can include Volatility Framework and Bulk Extractor. These tools allow for a basic analysis of data coming from RAM. This data includes, for example, active processes, credit card numbers, or passwords for login or user access authorization. The passwords themselves can be of great value in the acquisition of digital evidence, as they are able to allow access to protected resources such as files, devices, sites, password managers, or other key areas for the investigator. The aforementioned solutions from the operational memory area are based on simple signatures that, once the correct binary sequences are recognized, are able to display the information associated with the password (provided the software has the correct signature to search for). This process is rudimentary and insufficient to work effectively with a variety of data.

## 4. Overview of existing forensic tools used for RAM analysis

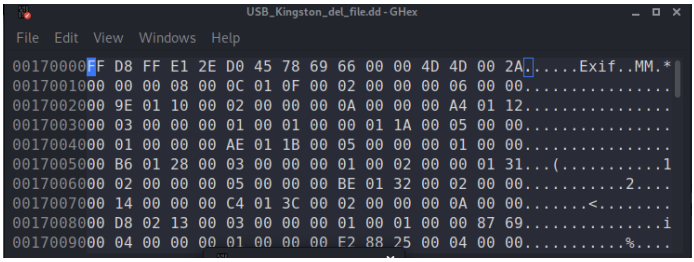
Sources of hints for possible passwords include all kinds of information found in digital media. Forensics analysts test various objects such as external storage media, computer disks, RAM, and information from other devices. Depending on the object under investigation, relevant information from the installed operating system can be revealed. Most operating systems have implemented logs that store data from the system operation, startup, shutdown, and error occurrences. Additionally, programs installed within the system may also have application logs that contain specific information about the application, such as changes made by the user. Each file that is installed or transferred to a device contains metadata, which includes information related to the last startup, creation, modification, and the user who made the interactions. Some files also include information related to the version or compilation date. Based on the file type and extension, we can infer the data it contains. Specific database files located in fixed locations can be associated with web browsing history, session-related information, or mail archives. During an investigation, it may

be necessary to analyze all available data to find artifacts that could impact the investigation. Analysts can use different types of tools to analyze selected objects and extract the most relevant information effectively. Individual profiled analysis tools can be used for this purpose. For example, here are some well-known and commonly used tools:

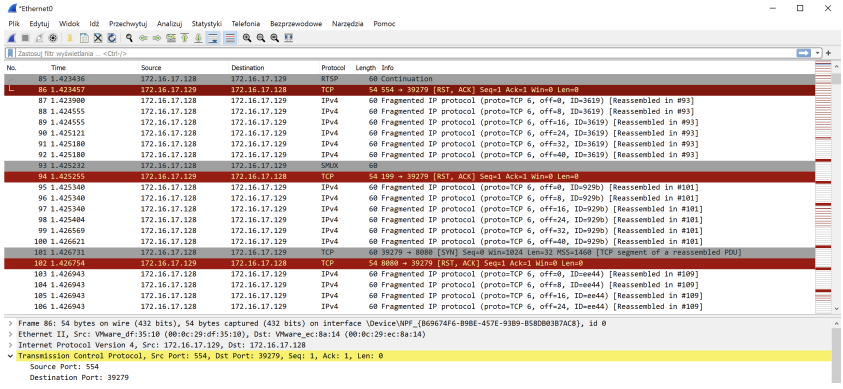
- **Volatility** is a powerful tool widely used for RAM analysis (in the form of memory dump files). It is a command-line tool and it is published as open source (also, the source code is available).



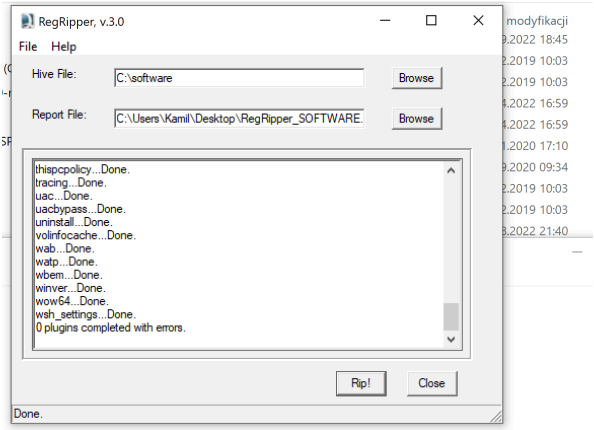

- **Hex editors** are a large group of tools for reading, manipulating, and writing binary data. This class of tools does not offer any special forensic functions; but, like any editor, it offers many tools for experienced users.




- **WireShark** is another powerful tool for analyzing and tracking network flows.



- **RegRipper** is a tool specifically designed for advanced analysis of Windows registers. It is also open-source software and provides two modes of work: GLU, and CLI.




- **The Sleuth Kit** is a collection of command lines tools used to inspect and recovery data from multiple file systems (NTFS, FAT, exFAT, HFS+, and Ext2/Ext3/Ext4).

On the other hand, it is becoming increasingly popular among analysts to use a tool that has multiple modules built in for various analyses. Such a model streamlines the analysis process and gives the possibility of the additional linking of some facts resulting from the analysis. Among such tools, we can include the following:

1. EnCase Forensic [5] is believed to be one of the most powerful and comprehensive digital forensics software suite developed by OpenText (https://www.opentext.com/products/forensic).
2. AccessData Forensic Toolkit (FTK) is another widely recognized and court-cited digital forensics software suite https://www.exterro.com/digital-forensics-software/forensic-toolkit.
3. Magnet AXIOM https://www.magnetforensics.com/ is a comprehensive and integrated digital forensics platform developed by Magnet Forensics. It is designed to help forensic examiners acquire, analyze, and report on digital evidence from a wide variety of sources, including mobile devices and IoT devices.

## 5. Traditional approach for RAM forensics

This paper aims to prove that analysis using a machine-learning model based on fastText solutions is a far more effective method than the traditional tools used to analyze memory to look for words that are likely to be passwords belonging to the user of that system. The aim of the study is to perform effective forensic analysis to extract valuable data sources.

## 5.1. RAM analysis for digital forensic

RAM is an essential component of many devices; it is used to store data and instructions that the computer can access at any given time. This memory consists of a number of cells, each with a unique address. Data exchange involves the processor pointing to the appropriate cell for modification. Modification includes (but is not limited to) operations such as addition, subtraction, division, and multiplication as well as logical operations (AND, OR, NOT, XOR). Note that all information stored in RAM is in binary form (see Fig. 1).


**Figure 1.** Example of raw RAM content

As RAM is a so-called volatile memory, the data stored on it is automatically erased as soon as power is removed, which can cause significant difficulties for investigations. In addition, RAM is dynamically managed by the installed operating system, which entails constant operations of modifying, writing, and deleting data.

Figure 2 shows the process of obtaining a snapshot from the computer's RAM. The process starts with the user running a RAM data-acquisition program. The tool, through proper interaction with the computer's operating system and drivers, manages access to the memory and hardware. The USB memory stick is the component that allows communication with the device to capture and store data in binary form. Most tools for this type of task interact with the operating system at a low level to access the memory. The captured data is then transferred to a USB stick to a specially prepared analysis station and subjected to inspection.

Currently, one of the most popular RAM-analysis tools is Volatility, which allows processes or system libraries to be extracted from a RAM dump for further analysis (e.g., for malware infection). Table 1 presents sample information retrieved from a RAM dump using Volatility.

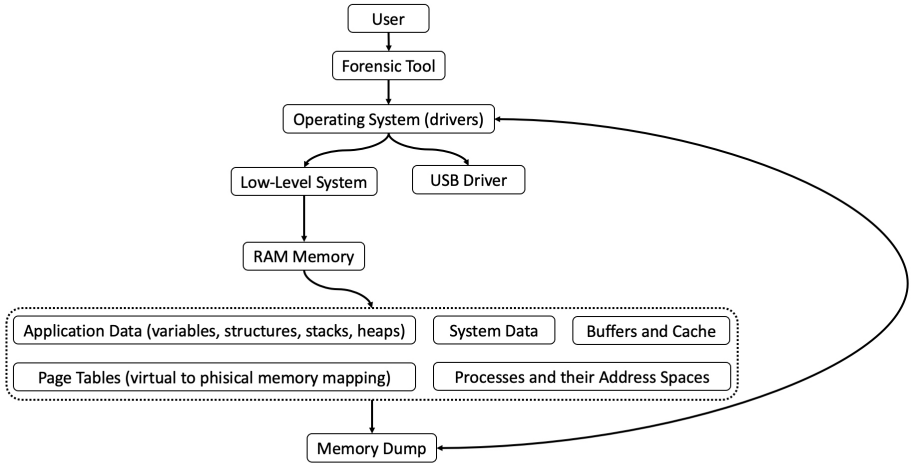



**Figure 2.** Workflow for dumping RAM from computer

**Table 1**
Output from Volatility tool – sample information retrieved from RAM dump

| Offset(P) | Name | PID | PPID | Thds | Hnds | Sess | Wow64 | Start | Exit |
|---|---|---|---|---|---|---|---|---|---|
| 0x1BA488040 | System | 4 | 0 | 193 | 0 | 0 | 0 | 2024-03-03 19:58:41 UTC+0000 | |
| 0x1BA50C080 | Registry | 140 | 4 | 4 | 0 | 0 | 0 | 2024-03-03 19:58:46 UTC+0000 | |
| 0x00000001842b7040 | smss.exe | 424 | 4 | 2 | 0 | 0 | 0 | 2024-03-03 19:58:41 UTC+0000 | |
| 0x190AA1400 | csrss.exe | 540 | 540 | 14 | 0 | 0 | 0 | 2024-03-03 19:58:43 UTC+0000 | |
| 0x0000000191b04800 | wininit.exe | 632 | 540 | 3 | 0 | 0 | 0 | 2024-03-03 19:58:44 UTC+0000 | |
| 0x0000000191421200 | csrss.exe | 640 | 620 | 17 | 0 | 1 | 0 | 2024-03-03 19:58:44 UTC+0000 | |
| 0x0000000194519800 | services.exe | 712 | 632 | 14 | 0 | 0 | 0 | 2024-03-03 19:58:46 UTC+0000 | |

**Table 1** cont.

| Offset(P) | Name | PID | PPID | Thds | Hnds | Sess | Wow64 | Start | Exit |
|---|---|---|---|---|---|---|---|---|---|
| 0x000000001945a0800 | lsass.exe | 732 | 632 | 13 | 0 | 0 | 0 | 2024-03-03 19:58:46 UTC+0000 | |
| 0x000000019419f080 | winlogon.exe | 784 | 620 | 7 | 0 | 1 | 0 | 2024-03-03 19:58:44 UTC+0000 | |

## 5.2. Testbed environment

The environment prepared for testing was created on a virtual machine (VMware platform) with Windows 10 installed (see Fig. 3). With this approach, it is possible to simulate a production environment on a physical host. A password-protected .rar file was created on the installed system (the file password was `Passw!DTrue`) with secret content. This article aims to summarize and compare the available analysis options, giving the chances of accessing the file based on a RAM (.mem) dump alone.

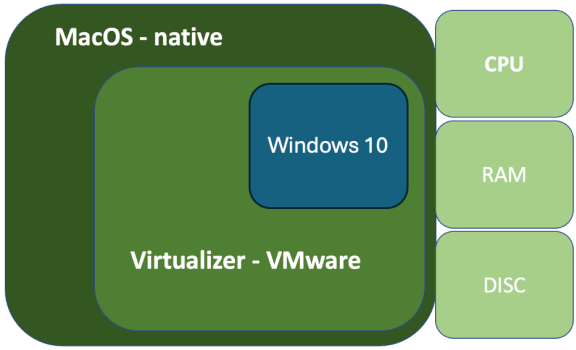



**Figure 3.** Testbed environment configuration – components can be easily replaced depending on target and available tools

At the moment, Windows 10 is still Microsoft's most popular operating system, holding 53% of the market as of March 2025; this is compared to 42.69% for Windows 11 (see Figure 4).

Furthermore, the analysis carried out does not focus solely on the system itself, as it is possible to further adapt the memory structure of Windows 11 at any time, optimizing it against Windows 10. Nevertheless, the proposed analysis approach does not limit the analyst to a single solution, as it is possible to analyze both Windows and Linux (no tests with macOS). This is due to the writing of data to the memory itself, a significant portion represented by the ASCII standard.

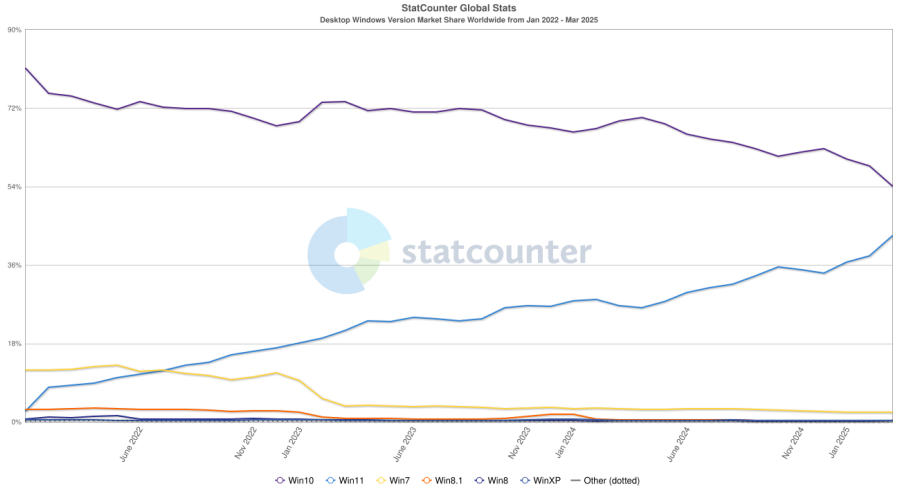



**Figure 4.** Windows 10 worldwide popularity according to Statcounter cite StatCounter2025

## 5.3. Application of standard/popular tools for retrieving password-related data

The use of Volatility showed that the tool could not locate or verify the necessary boot key (Hbootkey), which was used to decrypt the SAM database where Windows passwords are stored (see Figure 5).

```
┌──(kali㉿kali)-[~/Desktop/volatility3/volatility3]
└─$ python3 vol.py -f ~/Desktop/Win10_memdump.mem windows.hashdump

Volatility 3 Framework 2.7.2
Progress:  100.00               PDB scanning finished
User    rid     lmhash  nthash
WARNING  volatility3.plugins.windows.hashdump: Hbootkey is not valid


┌──(kali㉿kali)-[~/Desktop/volatility3/volatility3]
└─$ python3 vol.py -f ~/Desktop/Win10_memdump.mem windows.lsadump

Volatility 3 Framework 2.7.2
Progress:  100.00               PDB scanning finished
Key     Secret  Hex
WARNING  volatility3.plugins.windows.lsadump: Unable to find lsa key
```

**Figure 5.** Output from Volatility analyzing memory dumps in search for Hbootkey and LSA key

The memory dump may not have contained the required data for the program itself. The version of Windows or the state of the system at the time the dump was created may have affected the ability to obtain the Hbootkey. In addition, Volatility could not locate the LSA (Local Security Authority) key. The LSA key is essential for decrypting LSA secrets, which contain domain password caches and other sensitive information. As with the hashdump, the required data was not present in the memory dump. The information may have also suggested that the file was corrupt or there were compatibility issues with the Windows version. Each of these pieces of information proved how inflexible the above-mentioned tool was in analyzing RAM memory.

Another popular program that is used for analysis is Bulk_Extractor; despite finding quite a lot of interesting data for the investigation, this was also unable to retrieve the password for the .rar (see Figure 6).

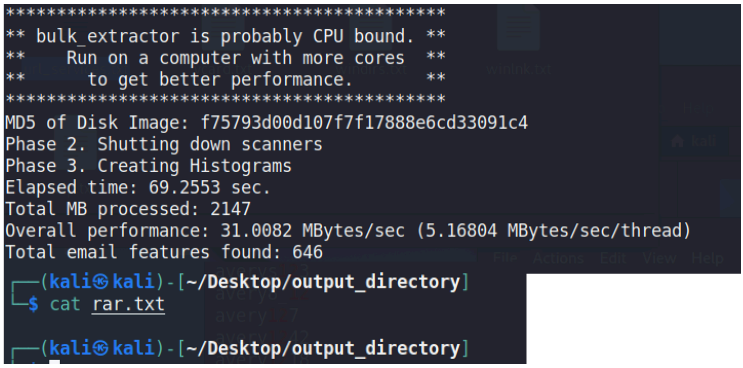



**Figure 6.** Output from `bulk_extractor` memory analyzing dumps in search for lsa key

## 6. RAM Forensics supported by machine-learning methods

In computer forensics, one of the important elements to be analyzed is a computer's RAM (Random Access Memory). Due to its technical characteristics, it is possible to extract various types of data from it, such as information about the operation of the system and active programs at the time when the so-called "memory dump" was created. This information unambiguously states what was happening on the analyzed station before it was switched off or in another way disconnected from the power source.

Since a typical user's activity is related to web-browsing, one of the basic pieces of information available for analysis today is data from a web page search history and the ability to retrieve image files from it; e.g., from the browser cache memory. Other elements include temporary files, which are not ultimately saved on the disk of the analyzed station; this data may prove extremely interesting and relevant to an ongoing investigation. User activity can be easily verified on the basis of open processes and information on the time at which a particular program was started and

closed. The most important aspect from the point of view of analysis is the unique property of being able to retrieve a profiled password database; therefore, the main objective of this article is to present a methodology designed for the acquisition of some data from the dumped operating memory of a computer station under study. We assume that any retrieved data will be used for generating dictionaries for dictionary cryptanalysis (e.g., using Hashcat). Due to the type of information stored in the memory, there is a certain probability of finding the correct password or passwords to access various files, devices, or services. The retrieved data can be also used as a hint for potential passwords – they can be used to produce a dictionary expanded from this data.

While many of the steps involved in analyzing memory dumps can be done with the tools presented in the earlier chapter, we developed our own tools in the Python language in this case. This maintains a high degree of flexibility for extracting and analyzing data found in RAM.

## 6.1. Preparing materials for analysis

A memory dump (see Figure 7) is a binary file that, when opened with basic editing tools, is unreadable by the user. If one does not intend to use specialized computer forensic tools, at least advanced knowledge of programming and data processing is required.

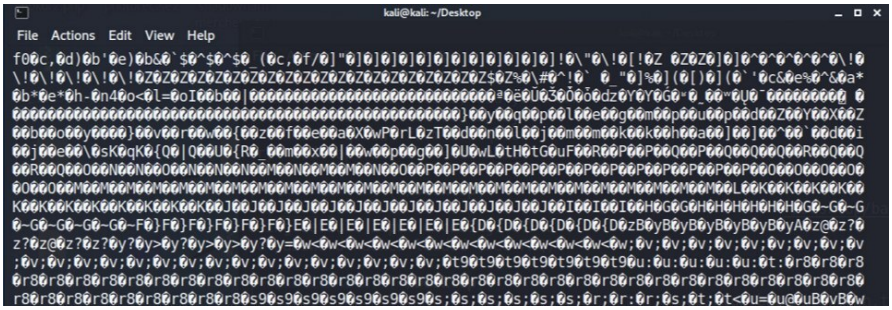



**Figure 7.** Example of string of characters located in memory dump

## 6.2. Cleaning materials before analysis

Prior to the analysis using the Machine-Learning model, the snapshot of the RAM must be cleaned. The memory dump contains a lot of information that is not useful for the operation of the models described. In order to prepare relevant data for analysis, it is crucial to extract readable and potentially relevant textual information from the raw memory data. The data consists of both textual and binary data. In addition, redundant information must be removed, although minimizing the modification of relevant information.The prepared file must be modified accordingly due to the format itself and the type of information stored. Here, we assume that only standard

ASCII characters are interesting – see Listing 1 (`allowed_characters`). Further development of the tool assumes that an increasing range of information in the memory dump file will be used in the analysis.

**Listing 1.** Python code for cleaning memory dump

```
with open(input_file_path, 'rb') as file:
    data = file.read().decode('encoding', errors='ignore')
filtered_data = ''.join(c for c in data if c in allowed_characters)

processed_lines = []

for line in filtered_data:
    words = line.split()
    new_words = [word for word in words if min_length
        <= len(word) <= max_length and not has_repetitions(word)]
    if new_words:
        new_line = '␣'.join(new_words) + '\n'
        processed_lines.append(new_line)
```

Removing non-ASCII characters from the memory dump file may not give a significant gain in memory savings and computation effort. In order to increase the efficiency of the analysis, filtering was applied to words of fewer than 7 characters and those exceeding 24 characters (`min_length` and `max_length` in Listing 1). This approach helps to focus on character strings that could potentially represent passwords while eliminating less likely character sets. In addition, the filtering function removes repeated words and those containing sequences of duplicate characters 'in a row,' which can often be the result of errors in the memory or irrelevant data. The imposed modifiable filtering rules minimize the amount of data needed for analysis, making it much easier to search patterns or undergo manual evaluation. The cleaned and reduced data provides a more useful form of information that can be used more effectively in the analysis process. Adjusting specific parameters can generate a larger or smaller database for analysis. The saved data should be in a format that can be used without too much trouble in the most popular tools – the purpose of which will be to use individual records from the plugged-in data set for a targeted designated attack.

The process of cleaning the data from the prepared memory dump is an important limiting element of the analysis process itself. Depending on the origin of the raw file from the RAM dump, the results can vary within a few, several, or dozens of megabytes, while raw files are up to 10 GB (see Figure 8).

Subsequent attempts on different files produced specific results. When analyzing a 1 GB RAM file, the cleaning program reduced its volume to 4.87 MB, which was only 0.0476% of the original file. The analysis of a 2.15 GB file looked similar; its cleaned version was only 5.62 MB, which was 0.0054% of the original file. Even in the

case of an almost-three-times-larger raw file of 6 GB, we instead obtained 29.5 MB of cleaned data, which represented 0.049% of the original file. It was noted that, each time larger files were analyzed, the output files did not increase dramatically.

```
Processing data: 100%|          | 1065147169/1065147169 [02:58<00:00, 5961731.85it/s]
File size of 'Win10_memdump.mem' before processing: 2048.00 MB
File size of 'cleared_Windows_10_2.15_GB.txt' after processing: 5.60 MB

Process finished with exit code 0
```

**Figure 8.** Cleansing memory dump file from unusable data.

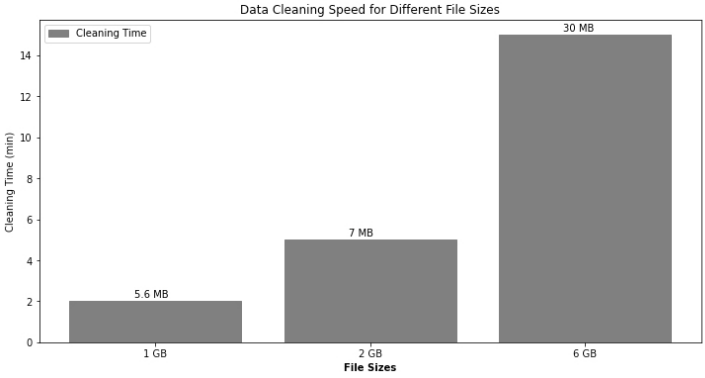



**Figure 9.** Cleaning computational efficiency – time needed to remove unusable data from memory dump

After cleaning the memory dump, usable text data must be extracted. Our method extracts data and expands into a dictionary of potential passwords. The dictionary was even larger than the memory dump (approximately by 70% – in experiments, the analysis of a 1.88 GB memory dump gave 3.21 GB dictionary files). The algorithm repeatedly writes a similar string of passwords that differ by one sliding character. Only such a way of generating single keys provides maximum efficiency and makes it possible to extract many potential passwords of a certain length. Unfortunately, this method has the significant disadvantage of low memory optimization, since potential passwords that have a low probability of occurrence in the analyzed area are also saved. There is a sure recipe for this in the form of limiting the number of occurrences of specific strings based on, for example, the currently used operating system (see Figure 10). Such an action would successfully minimize the size of the database itself by a fair amount.

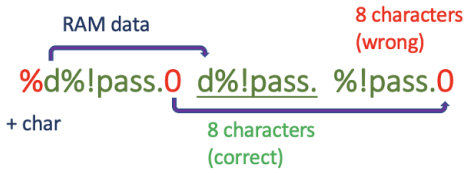



**Figure 10.** RAM data-sliding diagram

## 6.3. Machine-learning model for memory-dump analysis

The proposed model using machine-learning acting on the knowledge gained during the training analyzes a set of text phrases extracted from a memory dump and assesses their suitability as passwords (see processing flow in Figure 11).

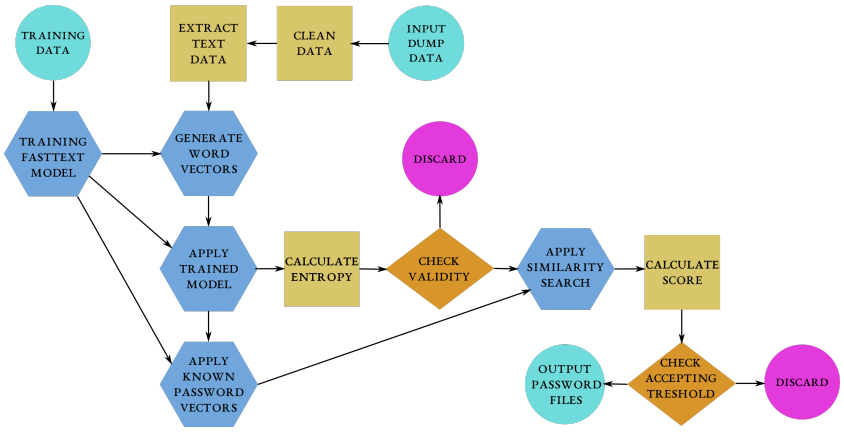



**Figure 11.** Workflow diagram for proposed tool

Here, we use the open-source tool fastText (https://fasttext.cc/) to analyze data acquired from a RAM dump. fastText, developed by Facebook AI Research, is a library designed for natural language processing (NLP). The model uses n-grams and subwords to increase robustness against rare words and spelling errors, which is particularly useful for inflected languages such as Polish. In contrast to deep machine learning, fastText enables effective training on large data sets with limited hardware resources. The Negative Sampling technique used in the model speeds up the training process by calculating gradients for only a small number of unrelated words, which significantly reduces the number of operations performed and shortens the time required for training.

The model was trained using lists available on the internet of passwords, such as the `rockyou.txt` file. At this stage of the work, such data was convenient to use because it allowed us to verify the validity of the method's assumptions: whether

the designed tool was able to find text phrases similar to the most frequently used passwords in the memory.

The model is capable of analyzing possible passwords for their similarity to those that are well-known most frequently chosen by users. The model effectively searches for similarity to learned vectors, determining the degree of similarity of a given password to those learned based on actual training data. The results of the study demonstrated the effectiveness and efficiency of using the model fastText in the analysis of secured RAM data, which may have important implications in computer forensics and for information security and data protection.

The most important part of the process was the preparation of the relevant training data and the subsequent correction of the relevant password factors in the `calculate_score` function (see Listing 2). Their incorrect selection could have resulted in an insufficient quality of the overall solution. Prior to the clean-up program, the most popular string for Windows 10 was the word 'Microsoft', which was directly or indirectly invoked more than 12,000 times (see Figure 12).

**Listing 2.** General structure of model

```
import fastText
clearData()
model = initialize()
model.get_word_vector(word)
    for i in range(0,N):
        model.calculate_score()
```

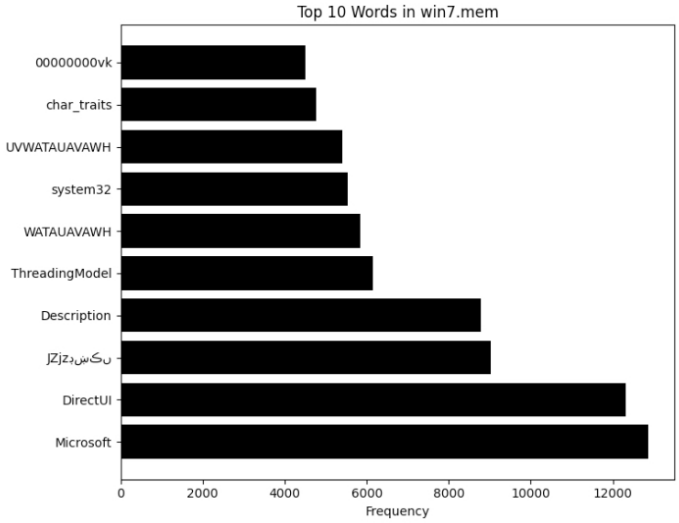



**Figure 12.** Top-10 phrases observed in memory dumps from Windows 10

Because the model is based on text data that has been prepared specifically for this task, the training database used (as a template) contains millions of records with sample passwords and provides a kind of sample of how users create their unique security features.

The next step in the processing workflow is a process that generates a word vector for each password using the `model.get_word_vector(word)` method. Next, fastText compares the word vectors with vectors of known passwords to assess how similar the word is to those already known. The fastText score (similarity of vectors) is used as part of the calculation of the final score for each password in the `calculate_score()` function. The proposed approach provides the possibility to assess the complexity of a password not only by its superficial features (such as length or use of special characters) but also by its semantic similarity to the training password base.

### 6.3.1. Evaluation of proposed solution

Having the dictionary, the dictionary attack can be performed; e.g., using Hashcat. This tool is intentionally optimized for high throughput processing using GPU (Graphic Processing Units). This type of computing platform is suitable for this type of attack since the GPU core units are able to calculate hundreds of hashes (digests) at the same time. Despite the fact that the computational power of GPUs grows very fast, the simplest form of attack – the brute-force attack – may fail when passwords are long enough (and use more types of characters than only letters and numbers). The only way to do this is to use various clues such as extracted text phrases from RAM. We will compare our method with a standard brute-force attack.

Using a trained and prepared model based on the fastText framework, it was noted that it can increase the efficiency of searching and profiling words that may have been used by a user as the correct password for a protected resource. This is done by processing single words into n-gram vectors and comparing them with vectors of known passwords using cosine similarity. Thus, the sum of these vectors can represent the correct word.

In order to estimate the effectiveness of the proposed solution, the same RAM dump file from a pre-prepared environment (Windows 10) was used. Due to the total lack of effectiveness of the popular programs used for RAM analysis (Volatility and Bulk_Extractor), the file was used as an input to three tools (see Fig. 13):

- 'ML_RAM': machine-learning model based on fastText;
- 'ram': analysis of cleaned memory dump file;
- 'brute force': classical brute-force attack using publicly available dictionaries.

Using the machine-learning model (fastText) 'ML_RAM' was able to access the protected file 10.8-times faster than when using a database based on filtered-only in-memory data. In the case of a brute-force attack, it was not possible to access the file in a reasonable amount of time. In order to be able to have a close look at how the model works on a smaller data set, a small data set of a dozen words was prepared (one of which was used to encrypt the file, with the rest coming from raw RAM data).

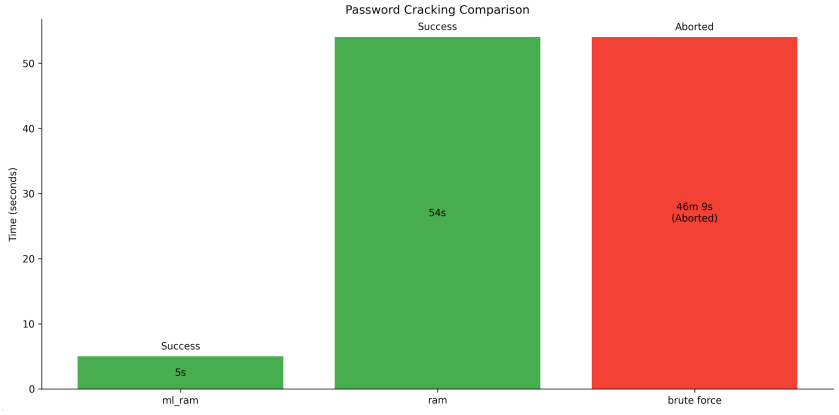



**Figure 13.** Performance of cracking passwords with Hashcat using dictionary provided by ML_RAM model, pure list of words found in RAM after cleaning, and direct brute-force attack; brute-force attack aborted after 46 minutes of computation

Using an advanced scoring algorithm based on machine learning, it was possible to obtain satisfactory results. Figure 14 shows the score for a few words found in the dumped RAM (after cleaning).

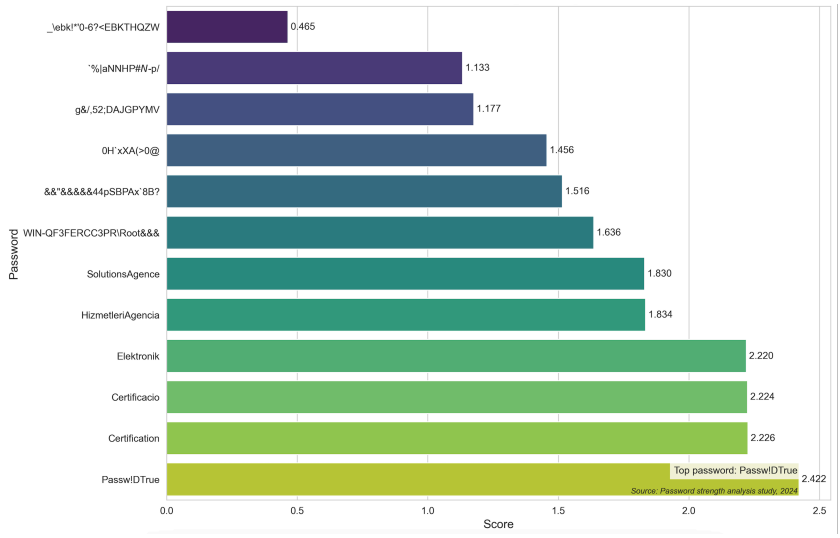



**Figure 14.** Scoring results for small sample set of words; highest score was assigned to text phrase '`Passw!DTrue`,' which was intentionally selected and used as password

The range of scores oscillated between 0.46 and 2.43. These results suggested a significant difference in the approach to the quality of the possible words.

The highest-scoring words were semantically meaningful, meaning that the algorithm definitely favored actual words (e.g., `+NofMicrosoft`) and its variants (e.g., `Passw!DTrue`). On the other hand, the lowest-scoring passwords were associated with a chaotic mixture of characters, special characters, and digits (e.g., '`_-\ebk*'0.6?<EBKTHOZW`').

This behavior was expected due to the material on the basis of which the fastText model was trained. A collection such as `rockyou.txt` contains relatively easy passwords based on natural language. The model therefore favors those words that are extracted from memory that resemble popular passwords.

As intended, our tool favored text phrases that were entirely or partly natural language words. It was clearly visible that passwords containing words or phrases that were understandable to humans (e.g., '`Passwd!DTrue`,' '`ServicedllnShield`,' '`ServiceMicrosoft`') generally received higher ratings than strings of characters that appeared to be random. In addition, there was no direct difference between the length of a password and its final rating. An example is the relatively short password '0H;xXAr*0@' (1.386), which received a higher rating than the longer '`_-\ebk*'0.6?<EBKTHOZW`' (0.46). Another important observation was that the presence of special characters did not guarantee a high score, as can be seen from the password '`h%R"Oif#S''` (0.8). Despite its complexity, it did not achieve the highest score. In order to test the method on more data, a set of 138 words was prepared that could be passwords.

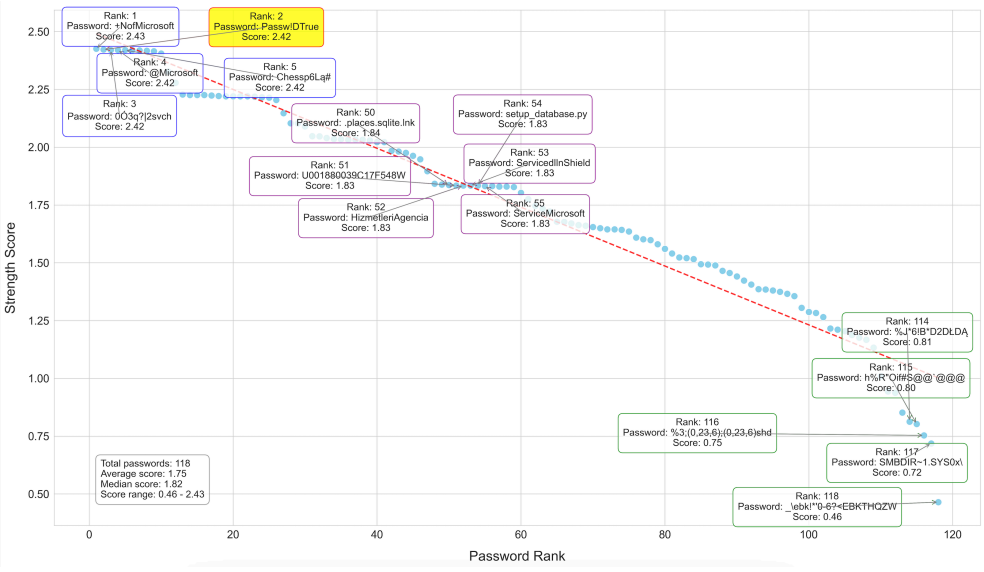



**Figure 15.** Strengths of ranked passwords

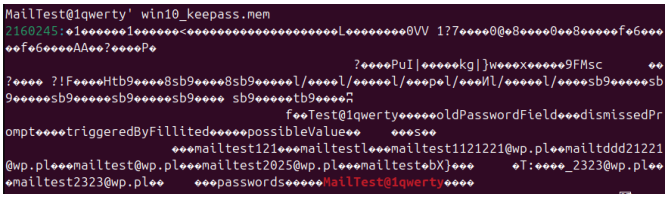
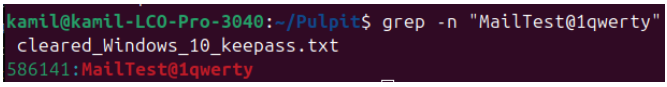
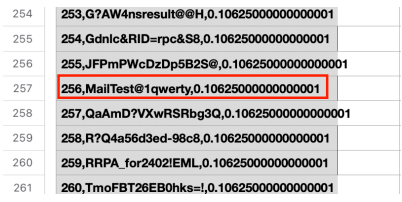
In summary, the analysis showed that the model used to assess passwords probably took into account not only character complexity but also factors such as word recognition, potential linguistic entropy, and possible semantic patterns. This measure may have reflected very sophisticated methods for assessing the correctness of passwords, going beyond traditional measures based solely on character length and variety. Ultimately, the best-scoring password for both the small and larger scales was '`Passw!DTrue`,' which was used as a password just before taking a snapshot.

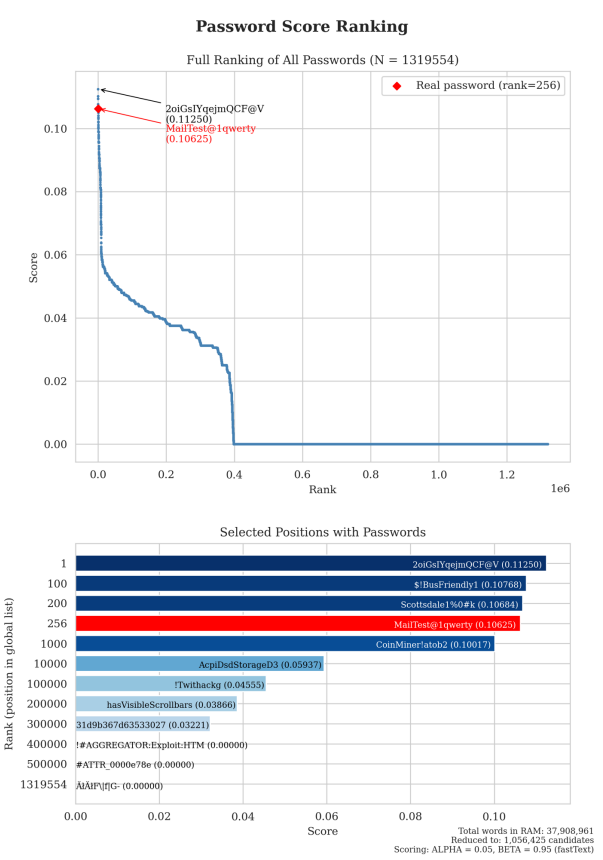
## 6.4. More case studies

In this chapter, we will present two more examples of analyzing a memory dump for passwords that should probably reside in the dump.

### 6.4.1. Keepass XC serving passwords for Gmail service

Keepass XC is a password manager; i.e., a tool for securely storing many different passwords for various services on a computer system. It helps when logging in to web services by transferring the password from its database to a form on a web page. A trace of the password's use remains in the computer's memory. The real password used in the Gmail account login method using the KeepassXC password manager was the word '`MailTest@1qwerty`.'

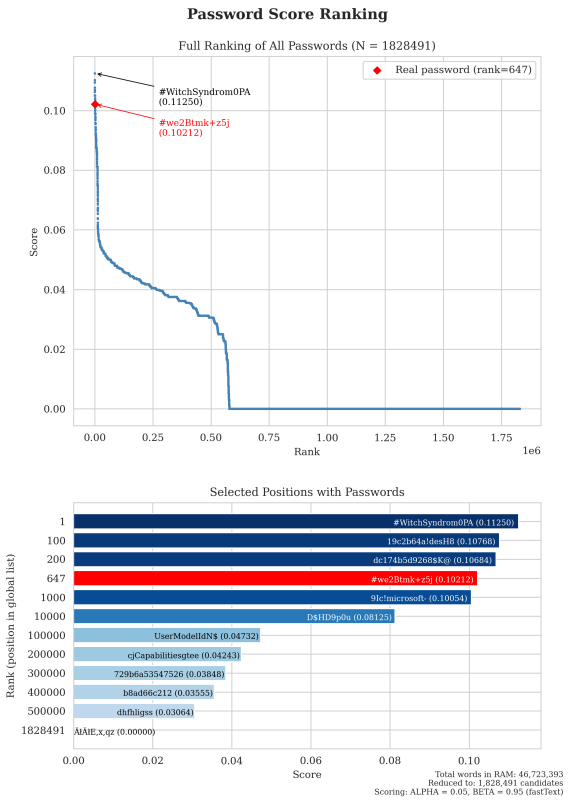
| | |
|---|---|
| Raw RAM dump contained 37,908,961 words |  |
| Password was located in memory dump on line 2,160,254 |  |
| Cleaning program reduced size of dump, and password was moved to position 586,141 |  |
| After processing by model, password was classified as 256 among most likely passwords |  |

| | |
|---|---|
| Password was among text phrases classified as being able to act as top-ranking passwords; as a result, potential attack could involve testing only a few hundred words instead of hundreds of millions |  |

### 6.4.2. Dropbox password stored by browser

Another example of retrieving a password from a memory dump was the case when the Dropbox password was automatically applied by a browser; the password was '`#we2Btmk+z5j`'.

| | |
|---|---|
| Original memory dump file contained 46,723,393 words | `kamil@MacBook-Pro-Kamil Program RAM v8.1 podob kosinusowe OK v2`<br>`% wc -w win10_dropbox_save.mem`<br>`46723393 win10_dropbox_save.mem` |
| Password was found in 2,775,408th position inside raw memory dump | `kamil@MacBook-Pro-Kamil Program RAM v8.1 podob kosinusowe OK v2`<br>`% grep -a -n '#we2Btmk+z5j' win10_dropbox_save.mem`<br><br>`2775408:#we2Btmk+z5j?????yGFILE0t??88??.`H8JΔ??kqΔ??kqΔ???f??x??`<br>`?? ??0hL??8JΔ??8JΔ??8JΔ??8JΔ?? index?00\r???  ?????yG`?;o?]e??`<br>`?_ 5'L?`?????$???? ?_M??`?  `?`<br>`J?P?; ???f?? ?_O?@ ????  ?<??? ?_?HRN?` |

| | |
|---|---|
| After cleaning: position 178,995 | `kamil@MacBook-Pro-Kamil USB % grep -n '#we2Btmk+z5j' cleared_Windows_10_dropbox.txt`<br>`178995:#we2Btmk+z5j` |
| Model classified password at position 647 in list of text phrases suspected to be password | 644,!d=12345678P,0.10212406251802891<br>645,!myPCsearch0,0.10212406251802891<br>646,#ToBase64StQ,0.10212406251802891<br>**647,#we2Btmk+z5j,0.10212406251802891**<br>648,$2!=pySherif,0.10212406251802891<br>649,$St0RPurchas,0.10212406251802891<br>650,$ac893057-PD,0.10212406251802891 |
| During classification using fastText model, real password obtained high rank, qualifying with high certainty for set of text phrases to be checked |  |

## 7. Conclusions

The process outlined in this paper is possible only to a certain extent. Note that RAM is ephemeral memory and stores its full data only when the station itself is running (continuous access to power). After switching off power, extracting data from volatile

RAM is impossible (or at least very difficult [7]). However, in digital forensics, any data retrieved from a computer system may be useful and may give insight into hidden information.

This article answers the question of whether it is possible to optimize dictionary attacks using data extracted from RAM. The research showed that a created database from memory had a high probability of having the correct password (or at least part of it). The tool used for generating a dictionary from RAM can be extended with an additional algorithm to expand the resource with random characters at the front, inside, and behind the extracted passwords.

This process will certainly greatly expand the available base; at the same time, it will increase the probability of obtaining the correct password value.

Compared to the types of attack studied, it can be unequivocally stated that it has many advantages such as the fact that the logging user of the analyzed station can repeatedly attempt to log in to various types of Internet services and password-protected files during the operation of the system. In addition, it should be noted that some users may use a single password in multiple places, which significantly increases the probability resulting from finding the right stack in the operating memory.

The described solution is effective but requires a certain amount of time. Depending on the parameter set during encryption, the time may not be within the acceptable framework of an investigation. Therefore, an additional solution that can improve the process of finding the correct password is the use of special dictionaries called rainbow arrays. Their use optimizes the process of indicating the correct constant necessary to decode the object under investigation.

The results are valuable for cybersecurity practices. In particular, they touch upon the security policies of companies and the approach to passwords of users themselves. It seems necessary to provide training and spread knowledge related to good practices and the possibilities of machine learning in the context of breaking simple security features chosen by users themselves. The analysis suggests the need for further in-depth research into the characteristics of data analysis, which could lead to the development of more-sophisticated algorithms for generating and evaluating passwords; this could be another tool in the hands of officers and cybercrime specialists.

The proposed method will be developed to create a generalized tool for computer forensics that will use machine-learning methods and contextual information when analyzing data (digital evidence); e.g., knowledge of a suspect's habits, or current usage guidelines/policies for specific resources (e.g., password policy). The target tool should support law enforcement by supplementing human reasoning prior to extracting valuable information from available data. The trained model of our tool will organize the relevant data, indicating to the human the priorities in the analysis.

## Acknowledgements

*We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support*

*within computational grant No. PLG/2024/017896. The research presented in this paper was partially financed using funds of the Polish Ministry of Science and Higher Education assigned to AGH University of Krakow.*

## References

[1] Ameri M.H., Blocki J., Zhou S.: Computationally data-independent memory hard functions, *arXiv preprint arXiv:191106790*, 2019.

[2] Biryukov A., Dinu D., Khovratovich D., Josefsson S.: Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications, RFC 9106, 2021. doi: 10.17487/RFC9106.

[3] Chen B.: *Memory-Hard Functions: When Theory Meets Practice*, Ph.D. thesis, UC Santa Barbara, 2019.

[4] Fleck A.: Cybercrime Expected To Skyrocket in Coming Years, 2024. https://www.statista.com/chart/28878/expected-cost-of-cybercrime-until-2027/.

[5] Garber L.: Encase: A case study in computer-forensic technology, *IEEE Computer Magazine January*, 2001.

[6] Gaur S., Chhikara R.: Memory forensics: tools and techniques, *Indian J Sci Technol*, vol. 9(48), pp. 1–12, 2016. doi: 10.17485/ijst/2016/v9i48/105851.

[7] Gupta K., Nisbet A.: Memory forensic data recovery utilising ram cooling methods, 2016.

[8] Halderman J.A., Schoen S.D., Heninger N., Clarkson W., Paul W., Calandrino J.A., Feldman A.J., Appelbaum J., Felten E.W.: Lest we remember: cold-boot attacks on encryption keys, *Communications of the ACM*, vol. 52(5), pp. 91–98, 2009. doi: 10.1145/1506409.1506429.

[9] Hausknecht K., Foit D., Burić J.: RAM data significance in digital forensics. In: *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1372–1375, IEEE, 2015. doi: 10.1109/mipro.2015.7160488.

[10] Hitaj B., Gasti P., Ateniese G., Perez-Cruz F.: PassGAN: A Deep Learning Approach for Password Guessing, 2019. doi: 10.1007/978-3-030-21568-2_11.

[11] Kävrestad J.: *Fundamentals of digital forensics*, Springer, 2020. doi: 10.1007/978-3-030-38954-3.

[12] Leimich P., Harrison J., Buchanan W.J.: A RAM triage methodology for Hadoop HDFS forensics, *Digital Investigation*, vol. 18, pp. 96–109, 2016. doi: 10.1016/j.diin.2016.07.003.

[13] Pasquini D., Cianfriglia M., Ateniese G., Bernaschi M.: Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries. In: *30th USENIX Security Symposium (USENIX Security 21)*, pp. 821–838, USENIX Association, 2021. https://www.usenix.org/conference/usenixsecurity21/presentation/pasquini.

[14] Pasquini D., Gangwal A., Ateniese G., Bernaschi M., Conti M.: Improving password guessing via representation learning. In: *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1382–1399, IEEE, 2021. doi: 10.1109/sp40001.2021.00016.

[15] Percival C., Josefsson S.: The scrypt Password-Based Key Derivation Function, RFC 7914, 2016. doi: 10.17487/RFC7914.

[16] Ravindra Sali V., Khanuja H.: RAM Forensics: The Analysis and Extraction of Malicious Processes from Memory Image Using GUI Based Memory Forensic Toolkit. In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 1–6, IEEE, 2018. doi: 10.1109/iccubea.2018.8697752.

[17] Su X., Larangeira M., Tanaka K.: How to prove work: with time or memory, *IEEE Access*, vol. 10, pp. 1192–1201, 2022. doi: 10.1109/access.2021.3138497.

[18] Thomas S., Sherly K., Dija S.: Extraction of memory forensic artifacts from windows 7 ram image. In: *2013 IEEE Conference on Information & Communication Technologies*, pp. 937–942, IEEE, 2013. doi: 10.1109/cict.2013.6558230.

[19] Yu F.: On Deep Learning in Password Guessing, a Survey, 2022.

[20] Zareen M.S., Waqar A., Aslam B.: Digital forensics: Latest challenges and response. In: *2013 2nd National Conference on Information Assurance (NCIA)*, pp. 21–29, 2013. doi: 10.1109/NCIA.2013.6725320.

## Affiliations

**Kamil Jurczyk**
AGH University of Krakow, al. Mickiewicza 30, 30-059, Krakow, Poland, jurczyk@agh.edu.pl

**Paweł Topa**
AGH University of Krakow, al. Mickiewicza 30, 30-059, Krakow, Poland, topa@agh.edu.pl

**Łukasz Faber**
AGH University of Krakow, al. Mickiewicza 30, 30-059, Krakow, Poland, faber@agh.edu.pl

**Received**: 11.12.2024
**Revised**: 20.02.2025
**Accepted**: 05.05.2025