Poster: Extracting Cryptographic Keys from Windows Live Processes

León Abascal and Ricardo J. Rodríguez[0000−0001−7982−0359]

Universidad de Zaragoza, Spain
{labascal, rjrodriguez}@unizar.es

Abstract. Cryptographic keys are a fundamental aspect of modern system security, but when compromised, they become a critical vulnerability, especially in ransomware attacks. Paradoxically, these keys must be available in memory at runtime to function, creating a unique opportunity for defensive tools. We introduce `KeyReaper`, an open-source tool designed to locate cryptographic keys in active Windows processes using advanced memory analysis. Unlike traditional approaches that rely on static memory dumps, `KeyReaper` performs dynamic analysis in real time, restricting the search to process heap memory to improve efficiency and accuracy. It employs robust key identification heuristics to minimize false positives and is designed for seamless integration with Endpoint Detection and Response systems. `KeyReaper` also encourages extensibility: its open-source nature allows researchers and practitioners to enhance its capabilities with custom key detection algorithms. We validated our approach through extensive experiments involving both proof-of-concept ransomware and real-world samples, demonstrating the effectiveness of key extraction and decryption success. Our tool provides a practical path to strengthening ransomware mitigation strategies.

Keywords: malware · Windows · digital forensics · cryptography

1 Motivation and Context

Ransomware has become one of the most significant threats in the cybersecurity landscape, as highlighted by multiple industry reports [5–7]. This type of malware primarily targets data availability [4], encrypting files or entire systems, and demanding a ransom in exchange for decryption keys.

The success of ransomware attacks depends largely on the use of strong cryptographic algorithms. However, operational errors in key management have led to the failure of some ransomware campaigns [1], including the use of flawed custom algorithms or poor key management practices [15]. In contrast, ransomware families such as `WannaCry` have employed robust and properly implemented encryption schemes that make recovery virtually impossible without the corresponding decryption key [2]. In this sense, the confidentiality and integrity of cryptographic keys are critical to the attack's effectiveness.

Cryptographic keys are typically generated using secure libraries (e.g., `Crypto++`) or system APIs (e.g., `CryptoAPI` for Windows) [2, 8]. Due to the

nature of the encryption process, these keys must reside in the ransomware's memory during its execution. This creates a limited, but interesting, window of opportunity to extract them, especially if their structure follows recognizable patterns. According to [3], key generation typically takes place on the victim's device, ensuring that the keys will be present at some point during the attack.

In response to this opportunity, we present `KeyReaper`, a novel tool designed to extract cryptographic keys from active Windows processes using advanced memory analysis techniques. Unlike existing approaches that rely on static memory dumps [10, 11, 15], our solution enables dynamic key extraction in real time by focusing on the process's heap memory, where such keys are most likely to reside during execution.

2 `KeyReaper`: System Design

In this section, we present the design of `KeyReaper`, our memory analysis tool developed to extract cryptographic keys from active Windows processes. It consists of three main modules and is designed to be used in an Endpoint Detection and Response workflow where a process has already been flagged as ransomware.

2.1 Workflow Description

Figure 1 illustrates the tool's workflow. The process begins with the ransomware actively running on a system, performing encryption operations and storing keys in memory. Let us clarify that `KeyReaper` does not include any ransomware detection mechanisms; rather, it is designed to operate after a process has already been flagged as malicious by an external security solution, such as an EDR system or antivirus engine. The tool consists of three main components that work sequentially to recover these keys.

The first module, *Process Capture*, pauses the execution of the ransomware process. This pause is essential to prevent future malicious behavior, including the potential destruction of encryption keys once the attack is complete. By stopping the process mid-execution, we ensure memory stability and a consistent state for extraction.

The second module, *Heap Extraction*, recovers the memory contents of the paused process. The tool specifically focuses on heap regions, as these are com-

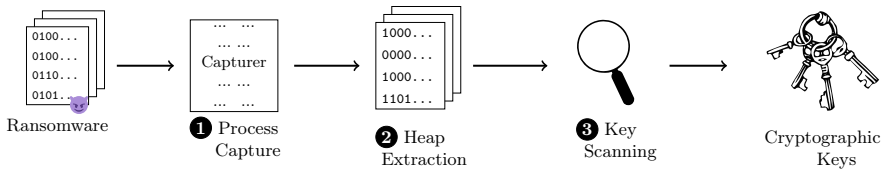


Fig. 1: System architecture and workflow of `KeyReaper`.

monly used for dynamic allocation of cryptographic keys and related metadata. Focusing on the heap improves performance and reduces noise.

The final stage, *Key Scanning*, analyzes the extracted memory using a series of interchangeable pattern recognition-based detection algorithms. These algorithms are designed to identify the structural signatures of cryptographic keys and enable the detection of symmetric and asymmetric keys. Upon successful identification, the keys can be extracted, allowing the victim's files to be decrypted without paying a ransom.

2.2 Design Highlights

This live memory-based approach offers several advantages over traditional post-incident forensics. By operating on active processes, the tool avoids issues such as memory paging, process termination, or data loss, which can occur when acquiring and analyzing a memory dump [10]. Its modular architecture also allows for the easy integration of additional analysis techniques, increasing adaptability to future ransomware variants. Additionally, the tool provides a command-line interface that supports options to pause, resume, or terminate processes, making it ideal for automated use in real-time incident response systems.

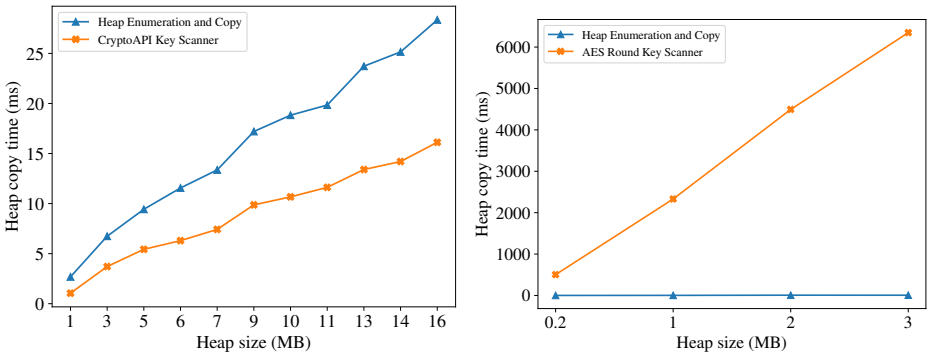
To facilitate key detection, `KeyReaper` includes several analysis modules based on well-established forensic techniques. One module implements the AES round key derivation method of [11], which reconstructs AES round keys from memory. This allows the detection of AES keys even when they are not explicitly present, by identifying characteristic patterns of the derived keys. While computationally intensive, it is still highly effective at detecting AES keys, which are widely used in modern ransomware families.

Another module is based on the technique originally presented in [15], which focuses on detecting keys generated using Microsoft's `CryptoAPI`. Although deprecated, `CryptoAPI` remains supported on all versions of Windows and is frequently used in ransomware. Our improved version of this method uses DLL injection and inter-process communication to extract key material from the target process with minimal disruption.

The tool's focus on heap memory is aligned with the goal of recovering dynamically generated keys, which only exist at runtime. Keys hardcoded in the binary or retrieved from a Command and Control server are beyond the scope of this tool and would require alternative recovery methods. By focusing on runtime memory, the tool achieves better performance and reduces false positives.

To accurately enumerate heap regions, `KeyReaper` uses the Process Environment Block (PEB). It also optionally employs pattern-based analysis of undocumented Windows heap structures, such as `_HEAP` and `_HEAP_SEGMENT`, available since Windows Vista. These structures provide reliable indicators of the boundaries and layout of heap segments, especially in fragmented memory.

To ensure responsiveness during live memory analysis, `KeyReaper` is implemented in pure C++ for optimal performance. The tool uses native Windows API calls to interact with remote processes, such as `NtSuspendProcess` to pause execution, `NtQueryInformationProcess` for process introspection,



(a) `CryptoAPI` key scanner. (b) Generic AES scanner.

Fig. 2: Comparison of performance for the two key scanning modules.

and `ReadProcessMemory` for safe memory access. Additional functions such as `GetProcAddress` are employed for dynamic symbol resolution. Memory regions are extracted using the `TitanEngine` library, which provides a high-level interface to low-level memory operations. For pattern matching within memory dumps, our implements the Boyer-Moore-Horspool algorithm, a fast and efficient method for substring searching in binary data.

3 Experimentation

To evaluate the reliability and effectiveness of `KeyReaper`, we conducted four types of complementary experiments. These experiments aim to assess: (i) the tool's reaction time, (ii) its resistance to false positives, (iii) the validity of the extracted keys, and (iv) its performance in real-world ransomware scenarios.

To evaluate reaction time, we developed a validation program that generates 100 cryptographic keys per run. We measured heap extraction and scanning times using `QueryPerformanceCounter`, repeating each configuration 1000 times with varying heap sizes. As shown in Figure 2a and Figure 2b, the AES scanner, although implementation-agnostic, is significantly slower. In contrast, the CryptoAPI scanner offers faster and more versatile performance.

To evaluate robustness against false positives, we introduced high-entropy noise into the heap using pseudo-random data in our validation program. We also evaluated the scanners with benign open-source software (e.g., `Audacity`, `Notepad++`) during typical usage sessions. In tests on synthetic and structured data, the tool consistently recorded a zero false positive rate, indicating high selectivity and reliability.

Next, we evaluated the accuracy of the extracted keys. Using the `CryptExportKey` [14] function, we exported the CryptoAPI-generated keys from our validation program. By parsing the exported key blobs and stripping their headers [13], we isolated the raw key material and compared it to the result gen-

erated by `KeyReaper`. This comparison was repeated for over 1,000 iterations, and in all cases, the keys matched exactly with no false detections.

Finally, we evaluated the tool in real-world ransomware scenarios using two prevalent malware samples: `Avaddon` and `WannaCry`. Both rely on CryptoAPI for key management. `Avaddon` generates a persistent AES session key for encryption [15] and continues monitoring the file system for new files, remaining active until manually terminated. WannaCry, on the other hand, creates an RSA key pair as a session key and uses it to encrypt the AES keys of each file [9]. Both samples were run in a controlled, isolated virtual environment (Windows 10 Pro, 4 GB RAM, 50 GB HDD). After confirming normal ransomware behavior (file encryption and displaying the ransom note), we paused execution at strategic times to extract the keys.

To confirm successful decryption, we placed a known plaintext file in the victim's environment, calculated its SHA256 hash, and resumed ransomware execution to enable encryption. After pausing the malware, we used the extracted key to decrypt the file and verified the result by comparing the SHA256 hash with the original.

These results confirm that its scanning modules reliably identify valid keys without false positives, and it performs well in both synthetic and real ransomware scenarios. Together, these findings demonstrate the tool's viability as a practical solution for dynamic key extraction during ransomware incidents.

4 Future Work

We plan to extend `KeyReaper` with support for additional cryptographic algorithms, including Salsa20. We are also developing mechanisms for detecting keys generated by popular libraries such as `OpenSSL` and `Crypto++`.

Future releases will include memory dump analysis for offline forensic investigations and benchmarking against existing similar solutions. Additionally, we aim to expand support beyond the NT heap to cover the segment heap introduced in Windows 10 [12].

Ongoing research on `CryptoAPI` will improve the detection of asymmetric keys, and we plan to evaluate the tool with legitimate cryptographic software for auditing and compliance testing. To improve robustness, we also plan to study heap manipulation and evasion techniques used by ransomware.

Acknowledgments. This research was supported in part by grant PID2023-151467OA-I00 (CRAPER), funded by MICIU/AEI/10.13039/501100011033 and by ERDF/EU, by grant TED2021-131115A-I00 (MIMFA), funded by MICIU/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR, and the University of Zaragoza, by grant *Proyecto Estratégico Ciberseguridad EINA UNIZAR*, funded by the Spanish National Cybersecurity Institute (INCIBE) and the European Union NextGenerationEU/PRTR, and by grant *Programa de Proyectos Estratégicos de Grupos de Investigación* (T21-23R), funded by the University, Industry and Innovation Department of the Aragonese Government. We used OpenAI's ChatGPT to improve the grammar, clarity, and coherence of the paper.

References

1. Aboud, M.A., Mariyappn, K.: Investigation of modern ransomware key generation methods: A Review. 2021 International Conference on Computer Communication and Informatics (ICCCI) pp. 1–5 (Jan 2021)
2. Bajpai, P., Enbody, R.: An empirical study of key generation in cryptographic ransomware. 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security) pp. 1–8 (Jun 2020)
3. Bajpai, P., Sood, A.K., Enbody, R.: A key-management-based taxonomy for Ransomware. 2018 APWG Symposium on Electronic Crime Research (eCrime) pp. 1–12 (May 2018)
4. Cawthra, J., Ekstrom, M., Lusty, L., Sexton, J., Sweetnam, J.: Data integrity: Identifying and protecting assets against ransomware and other destructive events. [Online; `https://csrc.nist.gov/pubs/sp/1800/25/final`] (Dec 2020), accessed on January 30, 2025
5. Cisco: Cisco Cyber Threat Trends Report: From Trojan Takeovers to Ransomware Roulette. [Online; `https://learn-cloudsecurity.cisco.com/umbrella-library/cyber-threat-trends-report`] (Jun 2024), accessed on April 9, 2025.
6. CrowdStrike: CrowdStrike 2025 Global Threat Report. [Online; `https://go.crowdstrike.com/rs/281-OBQ-266/images/CrowdStrikeGlobalThreatReport2025.pdf`] (Feb 2025), accessed on April 9, 2025.
7. European Union Agency for Cybersecurity: ENISA Threat Landscape 2024. [Online; `https://www.enisa.europa.eu/sites/default/files/2024-11/ENISA%20Threat%20Landscape%202024_0.pdf`] (Sep 2024), accessed on April 9, 2025.
8. Genç, Z.A., Lenzini, G., Ryan, P.Y.: Security analysis of key acquiring strategies used by cryptographic ransomware. In: Proceedings of the Central European Cybersecurity Conference 2018. CECC 2018, Association for Computing Machinery, New York, NY, USA (2018), `https://doi.org/10.1145/3277570.3277577`
9. Hsiao, S.C., Kao, D.Y.: The static analysis of wannacry ransomware. In: 2018 20th International Conference on Advanced Communication Technology (ICACT). pp. 153–158 (2018)
10. Ligh, M.H., Case, A., Levy, J., Walters, A.: The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. Wiley Publishing, 1st edn. (2014)
11. Maartmann-Moe, C., Thorkildsen, S.E., André Årnes: The persistence of memory: Forensic identification and extraction of cryptographic keys. Digital Investigation **6**, S132–S140 (2009), the Proceedings of the Ninth Annual DFRWS Conference
12. Mark Vincent Yason: Windows 10 Segment Heap Internals. [Online; `https://www.blackhat.com/docs/us-16/materials/us-16-Yason-Windows-10-Segment-Heap-Internals.pdf`] (2016), accessed on April 1, 2025
13. Microsoft Learn: BLOBHEADER structure. [Online; `https://learn.microsoft.com/en-us/windows/win32/api/wincrypt/ns-wincrypt-publickeystruc`] (Feb 2021), accessed on January 30, 2025
14. Microsoft Learn: CryptExportKey function. [Online; `https://learn.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptexportkey`] (Oct 2021), accessed on January 30, 2025
15. Yuste, J., Pastrana, S.: Avaddon ransomware: An in-depth analysis and decryption of infected systems. Computers & Security **109**, 102388 (2021)