Data Recovery via Memory Forensics Using System Call Monitoring

Ryo Higuchi, Hiroshi Inamura, and Shigemi Ishida
Future University Hakodate, Japan
{g2125068, inamura, ish}@fun.ac.jp

***Abstract*—Ransomware encryption attacks continue to cause severe damage, and backup-based restoration is often insufficient: even among victim organizations that maintain backups, only about 20% can fully restore their systems to the pre-attack state. Memory forensics-based key recovery is a backup-independent alternative, but recent ransomware suppresses in-memory key exposure through XOR masking or post-encryption key wiping, making the timing of acquisition critical. This paper proposes a memory acquisition method that monitors `read` and `write` system calls on decoy files to estimate the encryption window, and continuously dumps the heap and stack of the suspected process within that window. Using a pseudo-ransomware that masks its key outside encryption, the proposed method recovers files in 10/10 trials with a dump interval of 10 ms or shorter, while the write-triggered baseline fails in all trials. Continuous monitoring adds under 0.1 µs per file system call, and a 10 ms interval retains full recovery at 40% of the storage traffic of free-running acquisition.**

***Index Terms*—Ransomware, memory forensics, key recovery, system call**

I. Introduction

Ransomware encryption attacks have become a serious threat in recent years. According to the National Police Agency of Japan [1], 230 incidents against companies and organizations were reported in 2022, with a continuing upward trend since the second half of 2020. The common countermeasure is restoration from backups, yet although roughly 80% of victim organizations maintain backups, only about 20% of them fully restored their systems to the pre-attack state [1].

As a backup-independent alternative, memory-forensics-based key recovery has been studied [2], in which a memory dump of the running ransomware is analyzed to extract the encryption key. The critical limitation is timing: when the ransomware minimizes in-memory key exposure, by hiding the key with XOR [3] or wiping it immediately after encryption [4], identifying the right moment to dump becomes difficult. To improve the recovery rate, we propose a method that monitors the system calls issued by the ransomware to estimate when encryption occurs, and we quantify both the recovery rate and the overhead as a function of the dump frequency.

II. Proposed Method

We propose a memory acquisition method that estimates the short window in which the unmasked key resides in memory by monitoring `read` and `write` system calls on decoy files, and continuously dumps the heap and stack of the suspected process within that window so that at least one dump captures the key.

A. Read-triggered vs. Write-triggered Memory Dump

A ransomware that suppresses in-memory key exposure unmasks the plaintext key only immediately before encrypting a file and erases or remasks it immediately after the ciphertext is written, so the key resides in memory only during a short window between the `read` that loads the plaintext and the subsequent `write` that outputs the ciphertext. A *write-triggered* approach [5] dumps only after a `write` has been observed, sampling the process *after* encryption has finished, by which time the key has been wiped; it therefore cannot recover it in principle. Our *read-triggered* approach instead starts dumping as soon as a `read` on a decoy is detected and continues until the subsequent `write`, so the dumps fall *inside* the window in which the plaintext key is still present (Fig. 1).

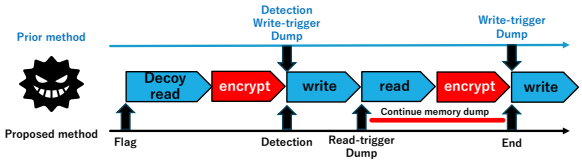

Fig. 1. Operation of the dump unit.

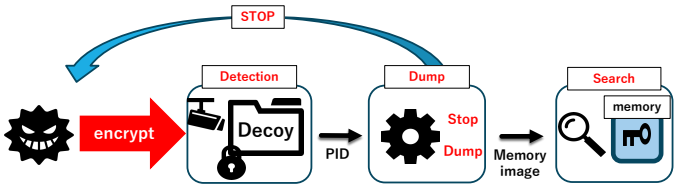

Fig. 2. Overview of the proposed method.

B. Method Overview

The proposed method, shown in Fig. 2, identifies the encryption window through system call monitoring and consists of three components. The **detection unit** places decoy files, which the user does not access under normal operation, in the file system and monitors `read` calls to them; a process that accesses a decoy is flagged and forwarded to the dump unit. The **dump unit** acquires memory dumps of the flagged process continuously from the `read` until the subsequent `write`, capturing the encryption window; to reduce dump size and acquisition latency, only the heap and stack regions are captured. The **search unit** extracts key candidates from each acquired image and uses them to decrypt the encrypted decoy; a candidate whose decryption matches the decoy plaintext is taken as the key and used to recover the remaining files.

Continuously dumping for every `read` increases system load and may affect other processes. We therefore quantify this cost separately for continuous monitoring and for incident-time acquisition, and examine how the dump interval trades recovery rate against overhead.

III. Evaluation

We evaluate the proposed method against a pseudo-ransomware, compare it with a canary-file baseline derived from Elastic Security Labs [5], and examine the trade-off between the dump interval and the recovery rate.

A. Pseudo-Ransomware and Experimental Setup

To emulate ransomware that exposes the key only during encryption, we implemented a pseudo-ransomware following evasion techniques reported for real-world samples. Its

This is the author's version of the work.
© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

TABLE I
RECOVERY RATE AND ACQUISITION COST (AVERAGED OVER 10 TRIALS)

Setting	Success	Gap (ms)	Written (MB)	CPU (%)
Baseline [5]	0/10	–	–	–
No monitoring	–	–	–	5.5
Detection only	–	–	–	4.2
Proposed (100 ms)	0/10	104.8	271	5.0
Proposed (50 ms)	0/10	53.0	522	5.9
Proposed (10 ms)	**10/10**	11.6	1565	9.3
Proposed (1 ms)	**10/10**	6.2	1897	10.3
Proposed (none)	**10/10**	0.03	3945	16.2

AES-256 key is XOR-masked immediately after generation and unmasked only just before encryption, reproducing the XOR-based concealment observed in real malware [3], and is overwritten with zeros as soon as encryption completes, reproducing ransomware that wipes the key after use [4]. Each iteration reads three to five files in a batch and waits a random interval, both reported for recent samples [4]. The random wait removes any periodicity from the windows; the batching, in contrast, lengthens each window relative to single-file encryption and is thus favorable to our method, a limitation we revisit in Section III-D.

Experiments ran on Ubuntu 22.04 LTS under Oracle VirtualBox on Windows 11, with one CPU core and 2 GB of memory allocated to the guest; monitoring used eBPF kprobes on `vfs_read` and `vfs_write`. The target consisted of 25 files of 10 MB encrypted with AES-256, plus one 2 KB decoy, which was assumed to be the first file encrypted so that the dump unit started immediately after the `read` on it was detected. As the baseline we implemented a write-triggered approach inspired by the canary-based mechanism of [5], which captures a single dump when a `write` to the decoy is detected. For the proposed method we evaluated five intervals between consecutive dumps: no interval, 1, 10, 50 and 100 ms. Each configuration was run for 10 independent attack trials.

B. Recovery Rate and Acquisition Cost

Table I reports, for each configuration, the number of successful trials and the cost of acquisition averaged over the 10 trials. The baseline failed in all trials, as expected from the timing argument above, while the proposed method recovered files in all trials at 10 ms or shorter and in none at 50 ms or longer. The number of key-bearing images grew monotonically as the interval shortened, and at 10 ms or shorter we reliably obtained at least one per trial.

The *gap*, measured from `SIGCONT` to the next `SIGSTOP`, is the target runtime between consecutive dumps and thus determines whether a dump lands inside the unmasked-key window. It follows the configured interval closely, confirming that the interval controls target runtime as intended. The cost of a cycle, in contrast, is set by the dump itself: each cycle writes roughly 45 MB of heap and stack, and its duration scales with that volume at about 3.5 ms per megabyte, so a cycle occupies about 100 ms of wall-clock time whatever interval is configured.

C. Continuous Monitoring Overhead

The detection unit runs whenever the system is up, so its cost cannot be justified by the urgency of an incident; and because the probes sit at the VFS layer, they fire on every file `read` and `write`, not only on decoy accesses. We measured the added cost per system call with a microbenchmark issuing page-cache-resident `pread`/`pwrite` calls, isolating the system-call path from disk I/O, running each configuration 10 times with 2×10^5 calls per run. Monitoring raised the median `read` latency from 372 to 468 ns and the `write` latency from 662 to 747 ns, reducing `read` throughput by 14.5%, below 0.1 µs per call. Consistently, detection running alongside the attack without dumping ("Detection only" in Table I) changed neither the encryption time nor the CPU utilization beyond run-to-run variation, confirming that the always-on component is deployable and the cost below is confined to the incident window.

D. Discussion

Since the target is frozen during the dump, the interval corresponds to target runtime rather than wall-clock time, as the measured gaps confirm. The interval is bounded from above by the unmasked-key window: recovery succeeded at a gap of 11.6 ms and failed at 53.0 ms, placing the window between these two values of target runtime, a tighter bound than the corresponding wall-clock times, so a 50 ms interval can step over it even though every encryption is bracketed by a `read` and a `write`. Below 10 ms, by contrast, the recovery rate is already saturated. The gap keeps shrinking as configured (11.6, 6.2 and 0.03 ms), so the target is indeed sampled more finely, yet all three settings recover 10/10. The additional dumps therefore buy no coverage while multiplying the cost, since each cycle costs about 100 ms of wall-clock time and 45 MB of writes regardless of the interval: free-running writes 3945 MB per incident against 1565 MB at 10 ms, at 16.2% CPU against 9.3%. The 10 ms setting is thus not merely sufficient but the least expensive of those achieving 10/10 recovery. The value is specific to this pseudo-ransomware and platform; the trade-off that determines it is not.

Freezing also converts part of this overhead into a benefit, slowing the attack by 14% at 10 ms and by 33% when free-running. The dominant remaining cost is storage, 1.5 GB per incident at the recommended setting, scaling with the heap size of the target. If a ransomware encrypts files one at a time, the window may shrink below what the dump duration can follow; because that duration is set by the volume written rather than by a fixed overhead, writing images asynchronously is the first target for improvement, along with a finer sweep between 10 and 50 ms and evaluation against more diverse samples.

IV. CONCLUSION

We proposed a memory-forensics-based recovery method that estimates the encryption window through system call monitoring and dumps within it. It achieved 10/10 recovery at a dump interval of 10 ms or shorter, whereas the write-triggered baseline failed in all trials. Continuous monitoring costs under 0.1 µs per file system call, and the incident-time cost (16.2% CPU and 3.9 GB written when free-running) can be cut to 9.3% and 1.5 GB at 10 ms without losing recovery. The interval is bounded above by the unmasked-key window, while below 10 ms the recovery rate saturates and further dumps only add cost; the numbers depend on the ransomware and platform, but this trade-off does not.

REFERENCES

[1] National Police Agency of Japan, Cyber Affairs Bureau, "Threats in cyberspace in 2022," 2023. [Online]. Available: https://www.npa.go.jp/english/bureau/cyber/document/threatsincyberspace2022.pdf

[2] P. Bajpai and R. Enbody, "Memory forensics against ransomware," in *Proc. 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, IEEE, 2020, pp. 1–8.

[3] K. Brezinski and K. Ferens, "Metamorphic malware and obfuscation: a survey of techniques, variants, and generation kits," *Security and Communication Networks*, vol. 2023, art. 8227751, 2023.

[4] C. Wang *et al.*, "RTCatch: a real-time detection method to combat evolving Rust language ransomware," SSRN preprint 5701272, Nov. 2025.

[5] S. Bitam and C. Alladoum, "Ransomware in the honeypot: how we capture keys with sticky canary files," Elastic Security Labs, Feb. 23, 2024. [Online]. Available: https://www.elastic.co/security-labs/ransomware-in-the-honeypot-how-we-capture-keys