Toward Automatically Generating User-specific Recovery Procedures after Windows Malware Infections

Jerre Starink
University of Twente
Enschede, The Netherlands
j.a.l.starink@utwente.nl

Cassie Wanjun Xu
TU Delft
Delft, The Netherlands
cassiexu0215@gmail.com

Andrea Continella
University of Twente
Enschede, The Netherlands
a.continella@utwente.nl

***Abstract*—Despite significant advancements in proactive malware detection and prevention, complete prevention of malware infiltration remains unattainable. Once malware is present on a system, it can make persistent changes that affect its stability, making user-specific recovery post-infection an important problem to address. Current solutions involve extensive monitoring to precisely pinpoint the changes that malware has made, which are impractical for home environments due to their high resource demands. This paper introduces a prototype for automatically generating user-specific malware recovery procedures that fully operates post-mortem. By leveraging forensic data collected on Windows by default, we replicate the original conditions under which the malware executed in a sandbox and automatically infer the exact system resources that the malware changed without imposing additional performance burdens on the user's machine. We test a prototype against 894 real-world malware samples and three real-world, environment-sensitive malware campaigns, and achieve a full recovery rate of 51.3% even with no additional monitoring enabled. We conclude by sharing insights on the importance of machine replication and sandbox configurability in future malware research.**

***Index Terms*—Malware, Recovery, Malicious Behaviors**

1. Introduction

Despite years of research put into malware analysis and infection prevention, we still face the threat of malware entering our computer systems [2].

One of the first lines of defense against malware on a user's machine is antivirus (AV) software. Typical AV inspects files statically for known threats (signatures) and performs lightweight system monitoring to look for anomalies. Over the past years, academic work has mainly focused on improving this *proactive* detection of malware [32]. While advances have been significant, it is impossible to fully prevent all malware from entering the system. Furthermore, even if AV detects malware retroactively (e.g., via updated signatures or heuristics), malware may still have had enough time to apply persistent changes to the system, such as dropping malicious files or changing critical settings that alter system behavior, potentially leaving the system in an unstable state.

With malware infections being imminent, *recovery* post-infection, that is, reverting any persistent system changes that the malware had applied, is crucial. However, research into this has been limited and require extensive monitoring or snapshots provided by End-Point Detection and Response (EDR) systems or similar [15], [16], [40]. This makes them an excellent solution for servers or enterprise environments. However, for the typical home user, this quickly becomes infeasible as full-system monitoring has massive computational power or disk space requirements, two features home systems typically lack.

An alternative approach is to analyze the malware post-mortem. After identification, the required recovery steps can be inferred by putting the malware in a sandbox and monitor its behavior from there instead [30], [31]. As the sandbox does not run continuously on the user machine, it does not sacrifice performance or disk space. However, a key limitation is that malware often features non-determinism. It may drop files with random names, change its behavior depending on the environment it runs in [1], or perform actions specific to a certain target audience [6], [21], [28]. This limits the *specificity* of the derived recovery steps, making it difficult to directly apply them to the user machine.

In this paper we extend the idea of post-mortem recovery step procedure generation by proposing work-in-progress prototype framework that fills in the missing details, allowing for *user-specific* recovery. Before running the sample in a sandbox, we first profile the user machine to replicate the conditions the malware originally ran under in a VM. We then collect forensic data on the user machine collected by the operating system *by default*, and automatically cross reference this with the sandbox report. This way, we automatically generate user-specific recovery steps that *requires no additional extra monitoring software* to be active and thus incurs zero overhead. We implement a prototype and measure its efficacy by testing it on 894 malware samples found in the wild, as well as three well-known malware campaigns that are known to be environment sensitive. We show that, even without any extra monitoring installed, we can identify and recover 51.3% of all persistent system changes completely, and demonstrate the necessity for a sandbox to be configured and customized to the user machine to make the full recovery procedure as complete as possible.

In short, we make the following contributions:

- We propose a prototype framework that leverages forensic data to replicate a machine in a sandbox to generate *user-specific* malware recovery proce-

dures post-infection *with zero overhead*.
- We implement our framework and test it against 894 malware samples found in the wild and three real-world, environment sensitive malware campaigns.
- We conclude by sharing insights on the importance of machine replication and sandbox configurability for future malware analysis research.

We release the data and source code of our prototype at https://github.com/utwente-scs/malware-remediation.

2. Background

Malware recovery post-infection involves reverting the *persistent changes* that malware has applied to the user machine that survive after neutralization. These may include created, deleted or changed files, as well as settings that have been modified by the malware, e.g., to survive reboots or compromise security [42]. The goal of malware recovery is to restore the machine to a stable state.

One approach to recover from infection is to revert the system to a known clean state, e.g., by using backups or performing a complete system reinstall. However, while this ensures all malicious persistent changes are removed, it also risks benign user changes to be lost as well. Furthermore, running back-ups periodically requires lots of disk space, and it is not always clear how far back a system has to be rolled back as malware can stay dormant for long periods of time [8], [20]. A more targeted recovery strategy is to identify files that match a known signature or exhibit some known malicious behavior. However, this may result in insufficient recovery, as signatures are often incomplete and may not recognize malicious intend or capture all the changes. In the ideal scenario, the machine is equipped with extensive behavioral analysis, e.g., provided by Endpoint Detection and Response (EDR), that continuously tracks all persistent changes on the system over time. However, this scenario is infeasible for home users, as continuous monitoring incurs significant space and runtime overhead, especially on machines with relatively limited storage or compute power. Finally, malware samples can be reverse engineered post-infection to infer its past behavior retroactively. However, doing this manually requires expert knowledge, and automating this comes with a challenge of malware often featuring non-determinism. Malware may be environment sensitive and change its behavior depending on the system state and configuration [1]. Furthermore, it may also randomize the names of dropped files or Registry keys, which complicates inferring the changes it specifically made and thus impedes generating *user-specific* recovery procedures.

3. Methodology

We now present our methodology to automatically generate *user-specific* recovery procedures for an infected machine. Our intended target audience is a home user that has a machine with limited computation and space capacity. In our threat model, we assume the infection on the user's machine has already happened. We also assume that the user was running a basic antivirus solution (e.g., Windows Defender) that was not able to detect malware

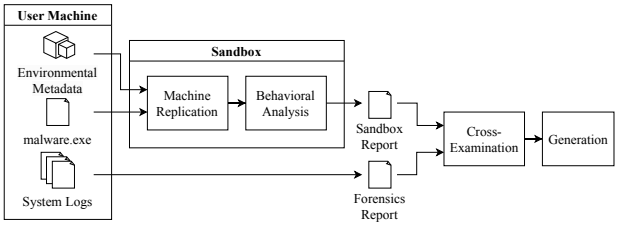


Figure 1: Overview of the approach. We use environmental data to replicate the user machine and system logs to generate user-specific recovery procedures.

successfully immediately, but identified the malware sample post-infection (e.g., via a static scan with an updated signature database, or via a behavioral anomaly heuristic at runtime), and neutralized it (e.g., by killing or quarantining it). Finally, we also assume no extra monitoring or backup services were set up beforehand (Section 2), and thus, we require a fully post-mortem approach. The end-goal is to generate directly applicable recovery steps that revert the persistent system changes made by the malware which directly impede normal operation of the user machine. Note that this does not include the recovery of encrypted or corrupted *user* files (which are often the target of e.g., Ransomware attacks). We intend to restore the system into a state that is clean of infections and stable again for daily use.

Since malware developers often obfuscate or pack their samples, static analysis is not feasible. Therefore, we resort to dynamic analysis where we run the sample a second time in a sandbox and recording all persistent changes. We use the key insight that modern operating systems feature many built-in sub systems that produce forensic data similar enough to the sandbox output to be able to cross-reference between them, solving the problem of non-determinism. Many of these sub systems are enabled by default and require no installation in advance.

Fig. 1 depicts our approach on a high level. We first replicate the user machine in a sandbox and run the sample in it to extract its persistent behavior. Then, we cross-reference the behavioral data with forensic evidence found on the user machine. Finally, we translate the changes made to the sandbox to changes on the user machine and generate recovery steps for each of them.

Machine Replication. We first construct a profile of the user machine by collecting environmental metadata (such as hardware information, user accounts, as well as the malware sample itself), and copying it into a Virtual Machine (VM). This has two main goals. Firstly, replicating the original conditions maximizes the chance the sample activating itself again a second time. Secondly, to generate user-specific recovery procedures, we need a trace that is representative to the user machine. Replication ensures that *system resources* (i.e., files and Registry keys) on the real machine will reside at a similar locations as in the VM, making cross-referencing more reliable.

Behavioral Analysis. We then use the VM as the base snapshot for a sandbox to rerun the sample in and automatically record the behavior that the sample exhibits during its execution. In particular, we specifically filter on system APIs involved in adding, modifying or removing

files and Registry keys, as these pertain to the persistent changes malware can make to the system.

Cross-Examination. After behavioral analysis, we automatically map the accessed, modified or deleted resources in the sandbox to forensic evidence found on the user machine. For modified and deleted system resources, this can be done by comparing their paths, as they will reside in the same location. However, for newly created files, a direct comparison of file paths is not always possible as file names the malware may drop files to may be randomized. The insight that we use here is that while file paths may be randomized, their *contents* are very likely going to be similar if not equal. A typical way to prove file equivalence is using a cryptographic hash algorithm (e.g., SHA256). However one single byte change in the file results in a completely different hash, rendering it unreliable for correlating files across systems that are near-identical. Therefore, instead we leverage fuzzy hash-based file contents similarity tests [19] to automatically map newly created files in the sandbox machine to files on the user machine, and define a *similarity threshold* parameter indicating the minimum required similarity score two files should have for them to be considered equivalent.

Generation. Finally, using the actions observed and the constructed system resource mapping, we automatically generate the final recovery steps. For every persistent change reported by the sandbox, we first invert all the recorded operations. This means for any file or Registry key created by the malware in the sandbox, we generate a file or Registry *deletion* step. Similarly, for modified or deleted resources, we generate a file or Registry *creation* step, where we pull the original data from a *reference system*, i.e., a fresh installation of the operating system which is known to be clean and stable. To ensure that the remediation steps are directly applicable to the user machine, we apply the resource location mapping constructed during cross-examination on the involved resources. Note that this is only feasible to do for resources that are present on the reference system. As stated before, our primary goal is removing malicious files and restoring system files and settings to ensure the machine is in a stable state, and does not include the recovery of corrupted user files (e.g., as a result of a Ransomware attack). Finally, we collect all the generated steps in a single report (e.g., JSON), which includes the resources and the appropriate actions to apply to them. These steps can then easily be translated to commands (e.g., a Powershell script) directly executable on the user machine.

4. System Architecture

We now describe our implementation of our prototype, filling in all technical details for implementing our methodology. We use CAPEv2 [29] as our sandbox as it features many behavioral analysis modules built in and allows for configuration of the VM. During Machine Replication, we copy the system architecture, system model, serial number and MAC address, and ensure the same specific installed version of the operating system is used in our VM. We configure the VM with the same username, user domain and hostname, to ensure file and Registry paths in our sandbox resemble the user machine. We copy locale settings (e.g., system language and timezones), as some malware campaigns target specific demographics or countries [6]. During Cross-Examination, we first determine the original time window in which the malware executed. Then, we refer to specific system services that are active by default, correlate their logs with the time window, and cross reference the records with the sandbox report.

4.1. Determining the Execution Time Window

To determine the time window in which the malware originally executed, we first turn to the Windows Event Log. Windows records process creation events in the Windows Event Log under Event ID 4688 (Process Creation) and 592 (Scheduled Task) [17]. This logging is enabled by default, and thus does not require installation or configuration of additional software. For duration, we move to *Prefetch* files. For every execution of a program, Windows creates a prefetch file to store data for faster loading of future instances [10]. These files also contain metadata, including total run times of the last running instance of the process. As they are generated by default, they can serve as forensic evidence that a process was active in the past. Prefetch files also include the process name and a hash of the file path, which we use to correlate them to the malware sample. Combining both the Event Log and Prefetch files gives us the complete time window the malware ran in.

4.2. Determining File System Changes

To get the files accessed by the malware, we need a record of file system changes on both systems. In the sandbox, we filter on the exact system APIs that specifically pertain to persistent file system modifications, including `NtWriteFile` and `NtDeleteFile`. For the user machine, we do not have additional monitoring and thus rely to file system forensic data indicating the same type of modifications have happened.

By default, Windows uses NTFS to manage files stored on the disk [24]. One feature of NTFS that is built-in and enabled by default, is the *USN Journal* [7]. This is a record that tracks file system modifications made to a volume, including file creation, writes and deletions. An example excerpt of a USN journal record is shown in Table 1. Here, we see a file modification, deletion and creation record. Although the USN Journal does not contain the process name or ID that was responsible for this modification, nor does it store the full path of the file, it does include the timestamp which we correlate to the malware's execution time window (Section 4.1), to make an initial over-approximation of all files accessed by the malware. Records also contain a sequence number which, together with Master File Table (MFT) reference number, can be mapped to a unique file identifier on an NTFS volume, and thus their full path. Depending on the update reason in the record, we proceed in the following manner:

Created Files. As mentioned before, given the non-determinism for file creation events in malware, we use fuzzy hash-based file content similarity tests to correlate dropped files across machines. For fuzzy hashing,

TABLE 1: Example record from `$UsnJrnl` with one file modification, one deletion and one creation event.

Name	MFT	Seq.	Timestamp	Reason
`foo.jcp`	103845	1	2025-01-21 23:33:40.78	DataOverwrite
`bar.tmp`	125581	12	2025-01-21 23:51:16.02	FileDelete
`baz.exe`	131432	5	2025-01-22 18:35:40.34	FileCreate

we chose Context Triggered Piecewise Hashes (CTPH) originally introduced by Kornblum et al. [19] as it is widely used by the community [26], [44]. To determine a suitable similarity threshold, we ran a small number of samples through our pipeline for which the created files were known, and took note of the similarity scores. We observed that the similarity scores were very distinctive: File pairs either had a score of $> 90\%$ if they were related, and $< 1\%$ when unrelated. Therefore, we use a minimum similarity threshold of 90% for two files to be considered equivalent.

Modified and Deleted Files. When malware modifies or deletes existing files, it is either targeting files made by the user themselves (e.g., to encrypt or corrupt) or files serving a specific purpose on the system (e.g., to install itself or compromise an existing service). As the recovery of user files is out of scope for this research (as this type of change can only be recovered by backups or self-healing file systems [9]), we only focus on the latter case. Evidently, content similarity tests provide limited insights when files are fully replaced and are impossible when files are deleted entirely. However, since system files are stored at specific file paths – contrary to the randomized names of dropped files – it is very likely that the malware modified or deleted the same system files with the same path in the sandbox as on the user's machine (provided the user machine is accurately replicated). Therefore, we directly cross reference paths with the records in the journal and add them to the recovery list if present on both systems.

4.3. Determining Registry Changes

Windows can keep track of Registry changes e.g., through event logs (Event ID 4657) [17] or in the form of transaction logs [23], [41]. To the best of our knowledge, none of these are enabled by default or are only recorded when multiple changes are bundled in a single transaction (which is unlikely for malware), and thus do not fit our use-case. However, similar to system files, Registry keys for specific purposes (e.g., adding persistence or turning off a certain service) usually has a specific path (with a few exceptions, see Section 6). We therefore consider the key path a good identifier to look up on the user machine, even without the verification from forensic evidence.

5. Preliminary Results

We now test our prototype to assess its capabilities in automatically composing recovery procedures from a machine post-infection.

5.1. Dataset and Setup

To verify that our prototype works, we assemble a dataset of 894 randomly selected samples from VirusTotal [43] flagged by at least three AV engines (as suggested by related work [49]) in 2020. We then use AVClass [37] to assign samples to family labels. The resulting set comprises samples from 128 distinct malware families (top 3: *wapomi*, *upatre* and *berbew*). We excluded a dominant ransomware family *virlock*, which took up half of the originally selected samples, as ransomware is out of scope for this research. Finally, the set also includes 55 samples that were not assigned a specific malware family.

To establish a *ground truth*, we first prepare a test machine with commonly used software (e.g., Google Chrome, Microsoft Office, VLC media player), that fulfills the role of a user machine. We then use the machine for normal daily activity for one month to introduce additional "wear and tear" artifacts [25], including browser history and normal system usage logs. Then, we run a sample on this machine for 5 minutes (as suggested by [20]), with an instance of *Sysmon* [39] and *Procmon* [38] actively running in the background. Note that we strictly run these tools to obtain ground truth traces and do not base our recovery procedures on these logs. We configure Sysmon to exclude most of the legitimate processes to reduce background noise, and let Procmon only log file-write events from non-system processes. After obtaining these logs, we generate the user-specific recovery procedures as described in Section 3, and compare the generated procedures with the ground truth traces to evaluate the effectiveness of our recovery system.

Following the community practices [33], [36], during the analysis, we make sure to deny potentially harmful traffic (e.g., spam) originating from both machines, and deploy our prototype on a separate sub-network where no production machines are connected. Additionally, after each analysis, we roll back the machines to a clean snapshot to revert side effects that malware might introduce. This also prevents potential denial of service attempts. Our setup was approved by our Ethics Committee.

5.2. Results

Among the 894 samples, we collected 741 (82.8%) results. 153 (17.1%) samples did not successfully execute (e.g., crashed or did not seem to be functional anymore). Out of the 741 working samples, 46 (5.16%) samples deleted their own Prefetch files, which renders our system unsuccessful directly. We reflect on potential solutions for this in Section 6. This leaves a total of 695 samples (77.7%) which were successfully handled by our prototype. Table 2 summarizes the detection and recovery rates of these samples. We define the recovery procedures as *complete* (fully recovered) if there are no malware-made changes left on the system. If the sample did not create or change files or Registry keys and our prototype correctly identified this (i.e., it did not incorrectly include additional files or Registry keys to wrongly "revert"), we also consider it a successful recovery. Our recovery system did not delete any legitimate resources and thus has zero false positives.

File System. Among the 741 active samples, 483 samples created new files on the system. Most of these samples only generated one file with a median of five files. There are 27 samples that created over 1000 files. For 155

TABLE 2: Observed recovery rates. *Full Recovery* indicates samples for which all objects were reverted. *No Recovery* indicates samples for which nothing was recovered.

	Full Recovery	No Recovery
Created File	53.3%	25.8%
Written File	98.0%	2.0%
Deleted File	99.5%	0.5%
Created Registry Key	99.2%	0.8%
Written Registry Key	94.1%	1.3%
Deleted registry Key	99.5%	0.5%
Total	51.3%	0.5%

samples (32.1%) we were able to identify and match files created in both the user machine and sandbox, of which 137 samples (28.3%) all the files were fully identified and recovered. Finally, for 191 samples (39.5%) we did not identify any file for recovery. We observe that file creation is the most variant behavior, and as such, the total recovery rate is mainly influenced by this.

For samples that created a large number of files (e.g., *sfone* family) we have a low identification rate. These mainly consist of randomly downloaded files which cannot be linked across separate instances of the malware. Additionally, we observe a trend where malware uses randomized names for new files and slightly modify their contents. From the total 1443 created files, 635 (44.0%) had a different name, and 626 (43.4%) had a similar CTPH ($> 90\%$ similarity) but not an identical SHA256 hash. This demonstrates the importance of our approach, as these files would not have been found otherwise.

We observed a large amount of file deletion activities from most samples in the dataset. When examining further, most of them can be attributed to malware trying to cover its tracks by cleaning up temporarily dropped files, emptying the recycle bin or deleting caches and logs related to its execution. Four samples deleted system files such as `iexplore.exe` and `notepad.exe`, and replaced them with their own version. In our recovery procedures, we did not identify them, as they were not reported by the sandbox analysis, likely due to CAPE sandbox limitations which we discuss in Section 6.

Finally, 15 samples modified existing system files, including `autorun.inf`, `system.ini`, `win.ini`. However, similar to the previously mentioned executable files, none of these were shown in the sandbox due to CAPE not picking up on these file changes.

Registry. Among the 741 samples, 166 explicitly tried to write data into Registry keys, of which 122 (73.5%) we were able to fully identify the changes, and for 10 samples we were not able to identify any written entries.

Most samples predominantly modified values under the `Run` and `RunOnce` keys. This is an expected result, as these are used to configure Windows to execute additional programs during system startup or user logon, and are thus are often used by malware to achieve persistence [42]. Note that for recovery for these keys, we only consider modification events (e.g., system calls such as `NtSetValueKey` and `NtDeleteKey`) and exclude creation events (i.e. `NtCreateKey`). We do this because `NtCreateKey` can be used for both creating a new key as well as opening an existing one. This is a reasonable compromise, as both keys are present by default on Windows 10, and have no lasting effect on the user's machine if this key is left empty.

We also observe that 15 samples created other Registry keys, of which 9 we can fully identify and recover the changes. These predominantly include modifying security policies and services (e.g., Windows Firewall), as well as adding binary payloads. If we inspect the remaining unidentified 7 samples, we observe that they try to register a system DLL under a unique identifier. As these identifiers can be randomized, it is impossible to trace back these keys on the user's machine just from the key path alone. We address this limitation in Section 6.

Finally, only four samples deleted keys. This is expected, as malware usually attempts to add extra (malicious) behavior, rather than removing (benign) behavior, and thus usually needs to only add or modify keys. Three samples removed the `AlternateShell` key (used to specify the first program to start when booting into Windows Safe Mode). The other removed a browser DLL and replaced it with its own DLL. None of them showed in the sandbox report, thus we are not able to recover them.

5.3. Case Studies

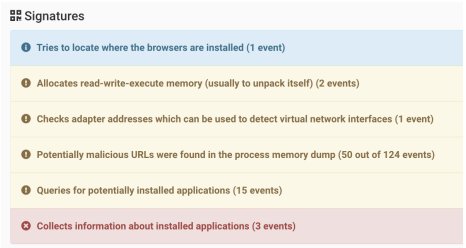
We now demonstrate the importance of machine replication by disabling it for three known malware campaigns.

Operation ShadowHammer. ShadowHammer is a high-profile supply chain attack from 2019 where the threat actors predetermined a set of roughly 600 hardcoded MAC addresses of the intended targets [14], [21]. When running a ShadowHammer sample[1] in our sandbox with its MAC address left random, we observe that it collects information but does not perform any action resulting in a persistent change (Figure 2a). However, changing the address to one of its intended targets (Figure 3b), we observe it also changes proxy settings (which would have to be reverted) and downloads additional files (which would have to be removed). This demonstrates that replicating specific hardware identifiers indeed results in a more complete picture of this malware's behavior, and thus allows us to generate a more complete recovery procedure.

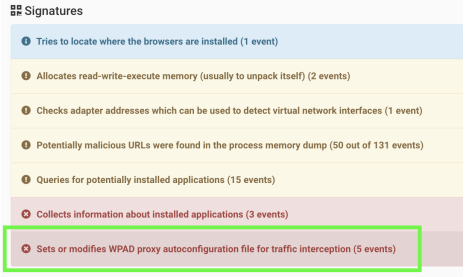
Raccoon Stealer. Raccoon is an info stealer that first appeared in 2019 that is known to fingerprint the host it is running on to determine if it is executing in a Russian or non-Russian environment and change its behavior accordingly [6]. In particular, we observe the malware[2] downloading payloads from a C2 server when residing in a UTC-5 American Eastern Time timezone and language set to English (Figure 3b), resulting in more of the malware's code to execute and make more persistent changes to the system than when executing the same sample within a Russian environment (Figure 3a). This shows localization settings are crucial in generating recovery procedures, or else behavior observed in a VM may not be representative.

OopsIE. OopsIE is a trojan developed by the OliRig APT group and targets Middle Eastern organization that only operates on the computer with Middle Eastern time zones (from UTC+2 to UTC+4) [28]. Similar to the case of Raccoon, we can replicate these localization settings in our

1. MD5 hash: 55a7aa5f0e52ba4d78c145811c830107
2. MD5 hash: 08a87b5af76a7d9f47d0bdd7453d77a4

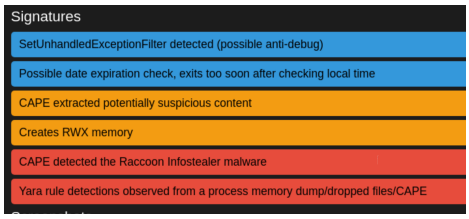


(a) Sandbox activity with randomized MAC address.

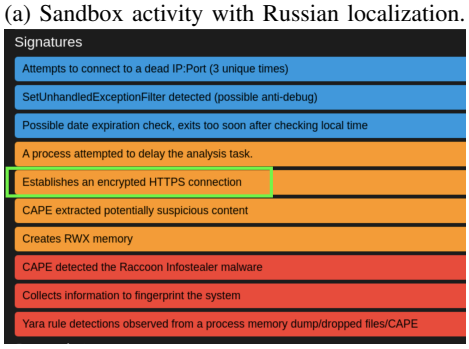


(b) Sandbox activity with targeted MAC address.

Figure 2: Operation ShadowHammer sample behavior.



(a) Sandbox activity with Russian localization.



(b) Sandbox activity with US localization.

Figure 3: Raccoon sample behavior.

sandbox and run a sample[3] for further analysis. However, even with this context replicated, we are still not fully capable of fully exposing the malware's behavior, due to additional detection evasion mechanisms that OopsIE implements. In particular, it checks for specific DLLs of the underlying virtualization software (e.g., VirtualBox or VMWare), and also tests for the existence of CPU temperature sensors and cooling fans [35]. While we can address the DLLs on the system (e.g., by removing or renaming them), unfortunately, CAPEv2 does not support simulating these specific hardware features yet. This further emphasizes the need for highly customizable VMs when trying to run a sample in a sandbox.

3. MD5 hash: a2402e839a5f159954a4afc695f8b265

5.4. Takeaways

Our results show an important insight. Even without any setup beforehand, with *built-in* features of Windows alone that are *enabled by default*, our prototype identifies and recovers a promising fraction of the persistent changes. The majority of modified and deleted system resources we can fully recover, and dropped files we can often identify and remove. The case studies further demonstrate that replication of user machine's environment is crucial in our pipeline, and calls for sandbox developers to make their VMs highly customizable.

6. Limitations

Our research does not come without limitations.

Limits of Forensic Evidence. To perform correct cross-examination between the sandbox and the user machine, our approach relies on USN Journals and Prefetch files being present on the user machine at analysis time. USN journal records have a limited time span, as they are constantly replaced with new data. Assuming 4 hours of system usage per day, the journal stores activity for up to two weeks [7], [27]. Our approach is therefore best applicable to relatively timely incident response scenarios. Additionally, from our experiments, we observed that 6% of samples deleted their own Prefetch files, which directly affect our prototype's success rate. However, there exist more forensic sources regarding program executions, including ShimCaches and AmCaches [34]. Combining these could make our prototype more resilient against malware intentionally deleting its own traces.

Registry Key Mapping Limitations. Our observations indicate that 44% of the created files and 7 Registry keys used a random name. While we can map files using content similarity tests, for Registry keys this is not feasible. Keys often have no contents or only contain small blobs of data (typically an integer or small string), making content similarity tests very prone to false positives. One improvement could be to generalize known paths into *patterns*. These patterns could also potentially be obtained semi-automatically, e.g., by running the sample multiple times and recognizing similarities in the observed paths [1].

Limited Metadata Coverage. Even though we carefully extract environmental metadata for replicating the user machine, it is possible that we missed components that could have impeded in malware reactivating itself a second time in our sandbox. A potential improvement to our system could be to make a full disk image of the user machine instead, to ensure the full file-system structure in our sandbox is equal to the system the malware originally ran under. Note that this still would require the extraction of auxiliary environmental metadata such as specific hardware identifiers and MAC addresses, as this is information that is not captured on a disk but is inherent to the hardware that the user machine is running on.

Recovery of User Files. While our prototype has been successful in remediating system resources and settings, it struggles to recover user files which can be the target of e.g., Ransomware attacks. While our primary intend has not been the recovery of encrypted or corrupted user

TABLE 3: Related work. *Specific* indicates user-specific generation directly applicable to the user machine. *Setup* indicates additional tooling is required. *Disk Space* and *Runtime* refer to incurred overheads.

Approach	Specific	Setup	Disk Space	Runtime
Hsu et al. [16]	Yes	Yes	0.3-13 MB/exec	107%
Goel et al. [15]	Yes	Yes	2 GB/day	7.4%
Torrellas et al. [40]	Yes	Yes	237.6 GB/day	4%
Paleari et al. [30]	No	No	None	0%
Our approach	**Yes**	**No**	**None**	**0%**

files, our system works port-mortem without making assumptions on the additional software running on the user machine. It could therefore also be used with no extra overhead costs on top of additional systems such as [9] and [4] that do offer recovery of user files post-attack.

Evasion. Despite our attempts to replicate the user's machine in our sandbox, malware could evade our prototype. For example, some samples rely on external factors such as an active C2 server, which may be offline or not very active. Additionally, even though CAPEv2 sandbox implements counter-measures and we add some of our own ourselves, malware may still detect our sandbox [3], [13]. An improvement would be to simulate more environment sensitive APIs (e.g., as suggested in [45], [47]), or switch to an agent-less sandbox (e.g., as suggested in [22]). Finally, samples might not have exhibited their behavior within our analysis timeout, although the majority of samples run within our limit [20].

7. Related Work

Automatic malware recovery post-infection has been explored in the past. In Table 3, we summarize previous work, their reported statistics, and how it compares to our approach.

Hsu et al. [16] proposed BACK TO THE FUTURE that divides processes into trusted and untrusted, and actively monitors all untrusted programs' operations. However, the user is responsible for identifying untrusted processes and give permission to them every time they attempt to make a persistent change, severely impeding normal use. Goel et al. [15] proposed TASER, which automatically identifies untrusted operations by searching in audit logs to determine the resources that were accessed (tainted) by the malware. However, next to the requirement of the presence of an auditing process, when legitimate operations depend on tainted operations, conflicts cannot be resolved which affects reliability. Others have also proposed VM-based solutions [5], [12], [18], [40], [46], where the execution of an entire machine is either restored to a known clean state, or replayed to infer which operations resulted in persistent changes. While their logging mechanism themselves impose low-performance overhead (typically not higher than 7%), virtualization is not free and incurs high storage overheads for its logs and snapshots, as well as overall runtime slowdowns. Besides, restoring to a previous snapshot is not a targeted remediation and files and Registry keys left untouched by the malware could be lost. Similar ideas have also been presented in [11], [48], None of these solutions, however, are really usable for a home user, as they also either heavily incur either runtime or storage overhead, and rely on additional software being installed on the user's machine. Our approach operates fully post-mortem and does not require any setup or specialized software running on the user's machine, removing any storage and runtime overhead during normal operation.

Paleari et al. [30] also worked on a framework to generate remediation steps of persistent changes applied by malware. Similar to our approach, the recovery procedures are generated entirely after the infection by running malware samples in a sandbox and thus incur no overhead on the user machine. However, they heavily rely on running the sample multiple times to generalize the potentially non-deterministic malicious behavior. In particular, they construct regular expressions for paths of files across individual runs based on common path prefixes and suffixes, in an attempt to map created files in the sandbox to files on the user machine. As the behavior of a sample is often highly sensitive to the environment it runs in [1], and file paths can be completely randomized (which we empirically showed in Section 5.2), such a file path generalization is not always reliable. This severely limits the specificity and thus direct applicability of the generated remediation steps. Instead, our approach actively replicates the user's environment to mimic the same conditions the malware ran under, and leverages file system forensics and fuzzy hashes to map created files by content and is fully independent of file paths, avoiding the need for such generalizations.

8. Conclusion

We presented a prototype to automatically generate user-specific malware recovery procedures that fully operates post-mortem. By replicating the original conditions the malware ran under in a VM, we can run samples in a sandbox a second time and cross-reference the observed behavior with forensic data present on the real machine. We empirically tested our prototype, and showed that, even without any additional monitoring software installed on the user machine, we can revert 51.3% of all the persistent changes the malware made. Finally, our study showed insights on the importance of machine replication and sandbox configurability in future malware research.

Acknowledgment

We would like to thank our reviewers for their valuable inputs. We also thank VirusTotal for granting us access to their academic malware sample dataset. This work has been supported by the SeReNity project, Grant No. cs.010, funded by The Netherlands Organisation for Scientific Research (NWO).

References

[1] Avllazagaj, E., Zhu, Z., Bilge, L., Balzarotti, D., Dumitras, T.: When Malware Changed Its Mind - An Empirical Study of Variable Program Behaviors in the Real World. In: Proceedings of the USENIX Security Symposium (2021)

[2] AVTest: Malware Statistics & Trends Report. https://www.av-test.org/en/statistics/malware/ (2025)

[3] Biondi, F., Given-Wilson, T., Legay, A., Puodzius, C., Quilbeuf, J.: Tutorial: An Overview of Malware Detection and Evasion Techniques. In: Proceedings of the International Symposium on Leveraging Applications of Formal Methods (ISoLA) (2018)

[4] Chen, N., Dafoe, J., Chen, B.: Poster: Data recovery from ransomware attacks via file system forensics and flash translation layer data extraction. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (2022)

[5] Choi, S.H., Park, K.W.: iContainer: Consecutive checkpointing with rapid resilience for immortal container-based services. In: Journal of Network and Computer Applications (2022)

[6] Cohen, B.: Raccoon: The Story of a Typical Infostealer (2020), https://www.cyberark.com/resources/threat-research-blog/raccoon-the-story-of-a-typical-infostealer

[7] Cohen, M.: The Windows USN Journal (2020), https://docs.velociraptor.app/blog/2020/2020-11-13-the-windows-usn-journal-f0c55c9010e/

[8] Comparetti, P.M., Salvaneschi, G., Kirda, E., Kolbitsch, C., Kruegel, C., Zanero, S.: Identifying Dormant Functionality in Malware Programs. In: Proceedings of the IEEE Symposium on Security and Privacy (S&P) (2010)

[9] Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barenghi, A., Zanero, S., Maggi, F.: Shieldfs: a self-healing, ransomware-aware filesystem. In: Proceedings of the 32nd annual conference on computer security applications. pp. 336–347 (2016)

[10] Duby, A., Taylor, T., Bloom, G., Zhuang, Y.: Detecting and classifying self-deleting windows malware using prefetch files. In: Proceedings of the IEEE Annual Computing and Communication Workshop and Conference (CCWC) (2022)

[11] Dunlap, G.W., King, S.T., Cinar, S., Basrai, M.A., Chen, P.M.: Revirt: Enabling intrusion analysis through virtual-machine logging and replay. In: Proceedings of the ACM SIGOPS Operating Systems Review (2002)

[12] Elbadawi, K., Al-Shaer, E.: TimeVM: a framework for online intrusion mitigation and fast recovery using multi-time-lag traffic replay. In: Proceedings of the International Symposium on Information, Computer, and Communications Security (ASIACCS) (2009)

[13] Galloro, N., Polino, M., Carminati, M., Continella, A., Zanero, S.: A Systematical and Longitudinal Study of Evasive Behaviors in Windows Malware. Computers & Security (2022)

[14] Gatlan, S.: MAC Addresses Targeted by the ASUS Supply Chain Attack Now Available (2019), https://www.bleepingcomputer.com/news/security/mac-addresses-targeted-by-the-asus-supply-chain-attack-now-available/

[15] Goel, A., Po, K., Farhadi, K., Li, Z., Lara, E.d.: The taser intrusion recovery system. In: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP) (2005)

[16] Hsu, F., Chen, H., Ristenpart, T., Li, J., Su, Z.: Back to the Future: A Framework for Automatic Malware Removal and System Repair. In: Proceedings of the Annual Computer Security Applications Conference (ACSAC) (2006)

[17] IT, U.: Windows security log encyclopedia (2023), https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/default.aspx

[18] Ko, R., Xiao, C., Onizuka, M., Huang, Y., Lin, Z.: Ultraverse: Efficient Retroactive Operation for Attack Recovery in Database Systems and Web Frameworks. In: arXiv preprint arXiv:2211.05327 (2022)

[19] Kornblum, J.: Identifying almost identical files using context triggered piecewise hashing. In: In Proceedings of the Digital Investigation (2006)

[20] Küchler, A., Mantovani, A., Han, Y., Bilge, L., Balzarotti, D.: Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes. In: Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS) (2021)

[21] Lab, K.: Operation ShadowHammer: new supply chain attack threatens hundreds of thousands of users worldwide (2019), https://www.kaspersky.com/about/press-releases/2019_operation-shadowhammer-new-supply-chain-attack

[22] Lengyel, T.K., Maresca, S., Payne, B.D., Webster, G.D., Vogl, S., Kiayias, A.: Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System. In: Proceedings of the Annual Computer Security Applications Conference (ACSAC) (2014)

[23] Microsoft: Working with transactions (2022), https://learn.microsoft.com/en-us/windows/win32/ktm/programming-model

[24] Microsoft: NTFS overview (2023), https://learn.microsoft.com/en-us/windows-server/storage/file-server/ntfs-overview

[25] Miramirkhani, N., Appini, M.P., Nikiforakis, N., Polychronakis, M.: Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In: Proceedings of the IEEE Symposium on Security and Privacy (S&P) (2017)

[26] Naik, N., Jenkins, P., Savage, N., Yang, L., Boongoen, T., Iam-On, N., Naik, K., Song, J.: Embedded yara rules: strengthening yara rules utilising fuzzy hashing and fuzzy rules for malware analysis. In: Proceedings of the Complex & Intelligent Systems (2021)

[27] Oh, J.: (2013), http://forensicinsight.org/wp-content/uploads/2013/07/F-INSIGHT-Advanced-UsnJrnl-Forensics-English.pdf

[28] O'Neill, P.H.: A well-known hacking group is getting better at evading detection (2018), https://cyberscoop.com/oopsie-oilrig-iran-evading-detection/

[29] O'Reilly, K., Brukhovetskyy, A.: CAPE: Malware Configuration And Payload Extraction (2024), https://github.com/kevoreilly/CAPEv2

[30] Paleari, Roberto and Martignoni, Lorenzo and Passerini, Emanuele and Davidson, Drew and Fredrikson, Matt and Giffin, Jon and Jha, Somesh: Automatic Generation of Remediation Procedures for Malware Infections. In: Proceedings of the USENIX Security Symposium (2010)

[31] Passerini, E., Paleari, R., Martignoni, L.: How Good Are Malware Detectors at Remediating Infected Systems? In: Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA) (2009)

[32] Quarta, D., Salvioni, F., Continella, A., Zanero, S.: Toward Systematically Exploring Antivirus Engines. In: Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA) (2018)

[33] Reidsma, D., van der Ham, J., Continella, A.: Operationalizing Cybersecurity Research Ethics Review: From Principles and Guidelines to Practice. In: Proceedings of the International Workshop on Ethics in Computer Security (EthiCS) (2023)

[34] Response, M.I.: New Microsoft Incident Response guide helps simplify cyberthreat investigations (2024), https://www.microsoft.com/en-us/security/blog/2024/04/23/new-microsoft-incident-response-guide-helps-simplify-cyberthreat-investigations/

[35] Robert Falcone, Bryan Lee, R.P.: OilRig targets a Middle Eastern Government and Adds Evasion Techniques to OopsIE (2018), https://unit42.paloaltonetworks.com/unit42-oilrig-targets-middle-eastern-government-adds-evasion-techniques-oopsie/

[36] Rossow, C., Dietrich, C.J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., Bos, H., Van Steen, M.: Prudent practices for designing malware experiments: Status quo and outlook. In: Proceedings of the IEEE Symposium on Security and Privacy (S&P) (2012)

[37] Sebastián, M., Rivera, R., Kotzias, P., Caballero, J.: AVclass: A Tool for Massive Malware Labeling. In: Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses (RAID). Cham (2016)

[38] SysInternals, M.: Process Monitor (2024), https://learn.microsoft.com/en-us/sysinternals/downloads/procmon

[39] SysInternals, M.: System Monitor (2024), https://learn.microsoft.com/en-us/sysinternals/downloads/procmon

[40] Torrellas, J., Oliveira, D.A.S.d., Crandall, J.R., Wassermann, G., Wu, S.F., Su, Z., Chong, F.T.: ExecRecorder: VM-based full-system replay for attack analysis and system recovery. In: Proceedings of the 1st workshop on Architectural and System Support for Improving Software Dependability (ASID) (2006)

[41] Via, D.: Digging up the past: Windows registry forensics revisited (2019), https://www.mandiant.com/resources/blog/digging-up-the-past-windows-registry-forensics-revisited

[42] Villalón-Huerta, A., Marco-Gisbert, H., Ripoll-Ripoll, I.: A Taxonomy for Threat Actors' Persistence Techniques. Computers & Security **121** (2022)

[43] VirusTotal: VirusTotal Malware Academic Dataset. https://www.virustotal.com/ (2020)

[44] VirusTotal: VirusTotal: ssdeep, CTPH hash of the file content (2020), https://developers.virustotal.com/reference/ssdeep

[45] VMray: Nowhere to Hide: Analyzing Environment-Sensitive Malware with Rewind (2016), https://www.vmray.com/cyber-security-blog/analyzing-environment-sensitive-malware/

[46] Webster, A., Eckenrod, R., Purtilo, J.: Fast and Service-preserving Recovery from Malware Infections Using CRIU. In: Proceedings of the USENIX Security Symposium (2018)

[47] Xu, Z., Zhang, J., Gu, G., Lin, Z.: Goldeneye: Efficiently and effectively unveiling malware's targeted environment. In: Proceedings of the Research in Attacks, Intrusions and Defenses (RAID) (2012)

[48] Zhu, N., Chiueh, T.c.: Design, implementation, and evaluation of repairable file service. In: Proceedings of the International Conference on Dependable Systems and Network (DSN) (2003)

[49] Zhu, S., Shi, J., Yang, L., Qin, B., Zhang, Z., Song, L., Wang, G.: Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In: Proceedings of the USENIX Security Symposium (2020)