Malware Family Classification via Residual Prefetch Artifacts

Adam Duby
United States Military Academy
adam.duby@westpoint.edu

Teryl Taylor
IBM Research
terylt@ibm.com

Yanyan Zhuang
University of Colorado Colorado Springs
yzhuang@uccs.edu

***Abstract*—Automated malware classification assigns unknown malware to known families. Most research in malware classification assumes that the defender has access to the malware for analysis. Unfortunately, malware can delete itself after execution. As a result, analysts are only left with digital residue, such as network logs or remnant artifacts of malware in memory or on the file system. In this paper, a novel malware classification method based on the Windows prefetch mechanism is presented and evaluated, enabling analysts to classify malware without a corresponding executable. The approach extracts features from Windows prefetch files, a file system artifact that contains historical process information such as loaded libraries and process dependencies. Results show that classification using these features with two different algorithms garnered F-Scores between 0.80 and 0.82, offering analysts a viable option for forensic analysis.**

I. Introduction

Incident response analysts use forensic techniques to analyze malware-based attacks. Unfortunately, manually analyzing malware is a costly and time consuming process that requires expert domain knowledge. Timely analysis is critical to mitigate attacks and exfiltration. Hence, there is a push towards automated malware classification techniques.

One of the popular malware classification techniques is to identify an unknown malware sample by its malware family. In order to do so, *features* from the malware are selected for comparison against a labeled dataset. Such features can include static information from the executable file (i.e, opcodes [22] or file header information [23]) or behavioral information extracted from a dynamic trace of the malware (i.e, API calls [14]). These features are often combined with machine learning techniques to classify a malware sample into a known family [21], [24]. Much of the current research on the topic assumes access to the original malware file. However, in many cases, analysts investigate attacks where the malware is no where to be found. For example, APT29, a threat group attributed with the 2020 SolarWinds compromise, is known to routinely delete its malware after completing a campaign [2].

Such self-deletion behavior forces analysts to find and analyze *residual forensic artifacts*, which are digital residue that may expose evidence about an attack [8]. For example, network logs and packet captures can expose network-related activities from an attack [19], while host-based artifacts can be extracted from memory and the file system [7]. Many of these forensics approaches are difficult to assess, as analysts either do not collect the proper logs, or do not have the expertise to properly analyze memory and file system data.

In this paper, we propose a Windows malware classification approach for cases where the original malware is unavailable. We leverage a Windows *loader* generated forensic artifact known as the Windows prefetch file. Prefetch files are created by the Windows Operating System after the first execution of an application and persist after an application is deleted. The intent of the prefetch file is to improve the performance of successive executions of the application by preloading dependencies such as dynamically linked libraries (DLLs), but the resource has been used for other purposes. For example, in digital forensics, prefetch files have traditionally been used as evidentiary artifacts of program execution [7]. Our work expands upon this practice to extract a feature set based on the libraries loaded by the malware during execution left behind in the prefetch file. A program's loaded libraries exposes insight into the program's behavior. As such, the *prefetch feature set* provides semantically valuable information for use in malware classification. One benefit of using this mechanism is that by default, it is enabled on all modern Window platforms; therefore, it is automatically available to the security analyst without being managed.

Our research focuses on malware classification to associate unknown malware with existing families. We do so using two approaches that use the prefetch feature set as input: an approach based on Jaccard similarity, and a technique based on a traditional machine learning ensemble classifier. Classifying malware into a known family encapsulates the malware's underlying behavior. As such, every malware instance in the family shares a common subset of semantics. For example, classifying a sample as WannaCry ransomware informs one that the sample exhibits behavior representative of WannaCry, e.g., file encryption and propagation via Server Message Block vulnerabilities [6]. If a common feature is removed from this subset, the malware's semantics are degraded because it fundamentally changes the behavior of the malware [16]. This is akin to a *monotonic increase constraint*, where the feature set of a variant may increase (i.e., features added), but the shared features common to the family cannot be removed [9] due to loss of functionality. We define the enforcement of a minimum feature space in malware classification as a *semantic preservation constraint*.

We propose a set-theoretic approach to enforcing semantic preservation constraints in malware classification using a

This work is supported in part by NSF grants OAC-2115134, OAC-1920462, and Colorado State Bill 18-086. The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Military Academy, the United States Army, the Department of Defense, or the United States Government.

prefetch feature set. Specifically, we extend Jaccard similarity such that the association with a family is penalized if an unknown malware does not contain the minimum features representative of the family. This technique has the added benefit of a fast filter that can quickly provide a reduced list of prospective malware families that meet the semantic preservation constraints. When evaluated using a dataset of malware from 48 families, our similarity-based classifier provides better performance over an ensemble classifier.

This paper makes the following contributions. First, we propose a feature set based on the Windows prefetch mechanism for malware family classification when a malware sample is not available; second, we describe an approach to enforce semantic preservation constraints in the feature set.

II. Background and Related Work

We now present related work on feature selection for malware classification, describe why malware self-deletion complicates such efforts, and present a technical overview of the Windows prefetch mechanism.

Malware Families. Malware classification can be used to associate malware with a known family. A malware *family* describes a grouping of malware that exhibit similar behavior to achieve similar effects. A family can describe commodity malware (e.g., njRat), or proprietary malware from an advanced persistent threat (e.g., BlackEnergy). Family classification exposes critical malware intelligence, informing post-attack decision making. For example, if a malware is classified as EternalRocks, then analysts know to hunt for suspicious Server Message Block traffic and scheduled tasks to detect and prevent successive EternalRocks activities.

Malware classification has been studied extensively in the past, with dynamic features known to provide the best classification performance. Dynamic features are extracted during program execution. Such features can include API call and system call information [3], [5], [14], function call graphs [17], or behavioral features such as resource interactions and mutexes [10], [13], [20]. Although dynamic features can overcome the limitations of static features, they require extensive feature extraction processes. Further, both static and dynamic techniques require access to the original malware.

Forensic techniques do not require an executable. Memory forensics has been shown to produce artifacts from process memory useful for clustering similar malware [11]; however, memory extraction is costly, and requires use of special tools and domain knowledge. Further, the Operating System reclaims memory after a process terminates, requiring analysts to extract memory while the process is running. Network forensics can be used to classify malware using network logs and packet analysis [19], while traffic pattern analysis can expose malware network behavior [12]; however, not all malware creates a network footprint and full packet capture for storage is impractical.

Malware Self-Deletion Behavior. Malware can delete itself after execution to evade detection and frustrate analysis. To estimate the pervasiveness of such behavior, we queried the MITRE ATT&CK framework [1] repository for threat actors and malware campaigns that are known to deploy self-deletion behavior. Our result shows that 29.8% of threat actors and 33.7% of malware campaigns have utilized self-deletion, suggesting that we cannot count on having malware available after an attack.

Prefetch Files. Our approach leverages the Windows prefetch file, a file system forensic artifact commonly used as evidence of program execution [7]. Prefetch files are used by the Windows OS to improve application start-up performance by loading application data into memory before it is demanded [18]. The OS loader generates these files for all recently executed applications. It monitors the references made by each process, such as DLLs, language and font files, child processes, etc., during load time (i.e., process startup) and the first ten seconds of runtime (i.e., process execution). The Windows superfetch service maps these references to the application's prefetch file. When the application is launched again, the loader checks the prefetch file to preload all the dependencies. Prefetch files have been utilized in malware detection [4], i.e., a *binary* classification. By contrast, we use the prefetch files to classify malware samples into a number of *different families*.

III. Approach

In this work, we consider a self-deleting malware and focus on residual forensic artifacts to find features suitable for malware family classification. More specifically, our approach extracts features from Windows loader prefetch files, which persist after program deletion. Further, we apply a novel set-theoretic technique to enforce semantic preservation constraints on the features. Our constraints serve as a filter to yield a list of prospective malware families associated with the unknown malware.

A. Feature Extraction and Preprocessing

To collect features, a malware is executed and its list of loaded libraries is extracted from the Windows' prefetch file. Preliminary analysis on our dataset (Section IV-A) revealed low variance DLLs that exist in every malware's prefetch feature set: `ntdll.dll`, `kernel32.dll`, and `kernalbase.dll`. These DLLs were filtered from the feature sets because they provide no semantically meaningful information. All applications load these DLLs to interface with the Windows OS. Further, a program's bitness (i.e., 32-bit versus 64-bit) is exposed in the prefetch file through the presence of Windows-on-Windows (WoW) subsystem DLLs: `wow64.dll`, `wow64win.dll`, and `wow64cpu.dll`. WoW enables a 32-bit process to execute on a 64-bit operating system. Since bitness is a byproduct of the underlying architecture and not consequential for program semantics, the WoW DLLs were removed from the feature set.

The remaining features can be represented using three ordering conventions: load-order, in-memory order, or unordered. Load-order is the order in which the files were loaded by the process, and is the order represented in the prefetch file. In-memory order is the order in which the libraries reside in virtual memory. Load-order can vary slightly between instances because the OS has a parallel loader, while in-memory order varies between executions due to address space layout randomization. Therefore, we represent the features as an unordered set to provide some normalization.

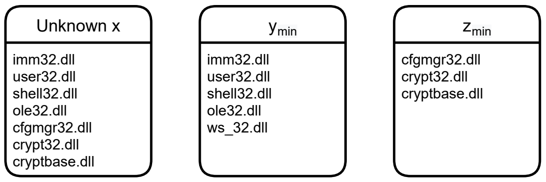

Fig. 1. Example Feature Sets.

B. Malware Classification using Jaccard Similarity

In order to classify malware into families, we want to measure the similarity of feature sets between samples. To do so, we use Jaccard similarity (JS), which measures the overlap of two sets into a score between 0 (no similarity) and 1 (exact match) and is calculated as follows. $\mathrm{JS}(\mathrm{A},\mathrm{B}) = \frac{|\mathrm{A}\cap\mathrm{B}|}{|\mathrm{A}\cup\mathrm{B}|}$. Unfortunately, a naive JS score does not guarantee a minimum feature space for malware classification (i.e., the semantic preservation constraint) [9].

Each malware within a family shares a common subset of semantics. We can represent this subset as a minimum feature set, or the intersection of all feature sets within the same family. This follows the intuition of a monotonic increase constraint for malware features [9]. Such constraints enforce the assumption that removing core features degrade program semantics. To show that Jaccard does not enforce constraints on set similarity consider an unknown malware's feature set, x. We want to assess the similarity of x with two known families, y and z. The minimum feature set for family y is represented as $y_{min} = \bigcap_{i=1}^{n} y_i$, and is the intersection of features across each instance y_i, while n is the total number of malware instances in family y; z_{min} for family z is computed similarly. The feature sets for x, y_{min}, and z_{min} are represented in Figure 1.

Here, Jaccard(x, y_{min}) $>$ Jaccard(x, z_{min}), suggesting that x may belong to family y. However, as shown in Figure 1, `ws_32.dll` $\notin x$ because x lacks Winsock TCP/IP networking functionality, which is one of the minimum semantic features associated with family y. By contrast, $x \supset z_{min}$, suggesting that x inherits the minimum semantic features of z. Therefore, we extend Jaccard similarity with a semantic preservation constraint.

1) Semantic Preservation Filter: We propose a set-theoretic approach to filtering the predicted family labels based on the intuition that malware families have a common subset of features. We name this the *semantic preservation filter*. Given the prefetch feature set of an unknown malware sample, the filter first yields a set of *prospects* that include malware families that meet the semantic preservation constraint. A classifier can then produce a *prediction*, which is the family that has the highest average similarity score with the unknown malware among the prospects.

Let Y be the set of malware families. For each malware family $y \in Y$, we find three feature-related signatures: the intersection of all features ($y_{min} = \bigcap_{i=1}^{n} y_i$, where n is the number of malware samples in y), the union of all features ($y_{max} = \bigcup_{i=1}^{n} y_i$), and the symmetric difference of the two ($y_{min} \triangle y_{max}$). If x is the set of features from an unknown malware instance, we first verify if x is a proper superset of y_{min}. If not, we disregard y as a prospective family because it violates semantic preservation. We also enforce a maximum constraint to ensure x does not contain excessive features not associated with a candidate family. Therefore, we check if the symmetric difference between x and y_{min} is a subset of the symmetric difference of y_{min} and y_{max}. If not, we disregard y because the feature set of x contains features not representative of the variance of family y. This approach yields a set Z of prospective families where $|Z| << |Y|$, filtering the search space for malware family classification. From Z, the family with the highest average pairwise Jaccard similarity with the unknown malware sample is predicted to be the family. The process is defined in Algorithm 1.

Algorithm 1: Family Classification.

```
/* Initializing variables */
$x$ : feature set from unknown sample
$Y$ : set of malware families, $y \in Y$
$y_{max} = \bigcup_{i=1}^{n} y_i$
$y_{min} = \bigcap_{i=1}^{n} y_i$
$Z = \{\}$        // $Z$ is the set of prospects

/* Use semantic preservation filter to find prospects for $x$ */
for $y$ in $Y$ do
    if ($x \supset y_{min}$) and ($x \triangle y_{min}$) $\subset$ ($y_{min} \triangle y_{max}$) then
        $Z.append(y)$;
    end
end

/* Find $z \in Z$ with the largest mean Jaccard similarity */
Prediction = max$\left\{\frac{|x \cap z|}{|x \cup z|}, z \in Z\right\}$
```

IV. Evaluation

We now describe the dataset (Section IV-A), then present our preliminary results (Section IV-B).

A. Dataset

Our dataset contains 4,126 malware samples from 48 malware families. The families capture a variety of malware behavior, including cyber espionage (Duqu 2.0), proxy-enabling click fraud (Nodersok), and self-propagating worms that install backdoors (EternalRocks). We executed each sample in a Windows 10 virtual machine and extracted their prefetch feature sets. The dataset was split into two: 75% used for training, and 25% used for testing.[1] The training set was used to find the y_{min} and y_{max} sets for each family.

B. Results

In this section, we first discuss our list of prospects from the semantic preservation filter, then present an analysis of our findings.

Semantic Preservation Constraints. We reduce the malware family search space by identifying prospect families. During testing, we found that the correct malware family was in the list of prospects 100% of the time. This filtered the original search space from 48 families to no more than 14 families. In some instances, the filter only yielded one prospect, in which case the need for further analysis is not required. An example is shown in Figure 2.

[1]We split the dataset based on compile stamps such that only the testing set contains the more recent variants from each family. This split avoids temporal bias [15], where a classifier is leaked future knowledge during training.

```
Prospects: [Ziyang, Bifrose, PotaoExpr, BeepService]
Prediction: Ziyang
```

Fig. 2. Sample output.

Analysis. After the semantic preservation filter yields a list of prospective families, the unknown malware is classified into the family from the list with the highest pairwise Jaccard similarity score. We analyze our results using F-Score, which is the harmonic mean of precision (p) and recall (r) such that $F_{score} = \frac{2*p*r}{p+r}$.

Our similarity-based classifier from Algorithm 1 provides an F-Score of 0.82. For comparison, we designed a traditional ensemble classifier using the prefetch features. The ensemble classifier uses a combination of popular machine learning algorithms (K-nearest-neighbor (KNN) and random forests (RF)). In addition to a slight performance hit ($F_{score} = 0.80$), the machine learning classifier does not guarantee semantic preservation. Further, a machine learning approach requires additional feature preprocessing to vectorize the feature space, and overhead in extending the training dataset. If a new family is added to a dataset, the approach from Algorithm 1 only needs to calculate a few simple set operations to obtain y_{min}, y_{max}, and the symmetric difference of the two. By contrast, the machine learning classifier approach requires retraining the entire model, which is a time consuming process. Our similarity-based approach offers more flexibility, and can be used with or without a malware sample, making it a valuable forensic tool.

V. Conclusions

This work proposes an approach to Windows malware family classification when a malware sample is unavailable. By extracting malware loaded library events from Windows' prefetch files, we obtained dependency information that can be used to build a similarity-based malware family classifier. Our results show that these features can be used to classify malware into families with high accuracy.

References

[1] ATT&CK. https://attack.mitre.org/.

[2] APT29. https://attack.mitre.org/groups/G0016/, 2021.

[3] M. Ahmadi, A. Sami, H. Rahimi, and B. Yadegari. Malware detection by behavioural sequential patterns. *Computer Fraud & Security*, 2013.

[4] B. Alsulami, A. Srinivasan, H. Dong, and S. Mancoridis. Lightweight behavioral malware detection for windows platforms. In *12th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2017.

[5] R. Canzanese, S. Mancoridis, and M. Kam. Run-time classification of malicious processes using system call analysis. In *10th IEEE Conference on Malicious and Unwanted Software (MALWARE)*, 2015.

[6] Q. Chen and R. A. Bridges. Automated behavioral analysis of malware: A case study of wannacry ransomware. In *16th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2017.

[7] A. Dimitriadis, N. Ivezic, B. Kulvatunyou, and I. Mavridis. D4i-digital forensics framework for reviewing and investigating cyber attacks. *Array*, 2020.

[8] V. S. Harichandran, D. Walnycky, I. Baggili, and F. Breitinger. Cufa: A more formal definition for digital forensic artifacts. *Digital Investigation*, 2016.

[9] Í. Íncer Romeo, M. Theodorides, S. Afroz, and D. Wagner. Adversarially robust malware detection using monotonic classification. In *4th ACM International Workshop on Security and Privacy Analytics (IWSPA)*, 2018.

[10] C. Jindal, C. Salls, H. Aghakhani, K. Long, C. Kruegel, and G. Vigna. Neurlux: dynamic malware analysis without feature engineering. In *35th ACM Annual Computer Security Applications Conference (ACSAC)*, 2019.

[11] M. A. Kumara and C. Jaidhar. Leveraging virtual machine introspection with memory forensics to detect and characterize unknown malware using machine learning techniques at hypervisor. *Digital Investigation*, 2017.

[12] K. Makhlouf. Finding a needle in a haystack: The traffic analysis version. *Proceedings on Privacy Enhancing Technologies*, 2019.

[13] R. Mosli, R. Li, B. Yuan, and Y. Pan. A behavior-based approach for malware detection. In *IFIP International Conference on Digital Forensics*. Springer, 2017.

[14] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[15] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium*, 2019.

[16] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *41st IEEE Symposium on Security and Privacy (S&P)*, 2020.

[17] C. Puodzius, O. Zendra, A. Heuser, and L. Noureddine. Accurate and robust malware analysis through similarity of external calls dependency graphs (ecdg). In *The 16th International Conference on Availability, Reliability and Security*, 2021.

[18] N. Shashidhar and D. Novak. Digital forensic analysis on prefetch files. *International Journal of Information Security Science*, 2015.

[19] L. F. Sikos. Packet analysis for network forensics: A comprehensive survey. *Forensic Science International: Digital Investigation*, 2020.

[20] J. Stiborek, T. Pevnỳ, and M. Rehák. Multiple instance learning for malware classification. *Expert Systems with Applications*, 2018.

[21] D. Ucci, L. Aniello, and R. Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 2019.

[22] J. Upchurch and X. Zhou. Malware provenance: Code reuse detection in malicious software at scale. In *11th IEEE Conference on Malicious and Unwanted Software (MALWARE)*, 2016.

[23] G. D. Webster, B. Kolosnjaji, C. von Pentz, J. Kirsch, Z. D. Hanif, A. Zarras, and C. Eckert. Finding the Needle: A Study of the PE32 Rich Header and Respective Malware Triage. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. 2017.

[24] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, 2017.