Metamorphic Malware Behavior Analysis using Sequential Pattern Mining

M. Saqib Nawaz[1[0000−0001−9856−2885]], Philippe Fournier-Viger[1[0000−0002−7680−9899]], M. Zohaib Nawaz[2[0000−0001−9205−912X]], Guoting Chen[3[0000−0003−2072−1588]], and Youxi Wu[4[0000−0001−5314−3468]]

[1]School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), Shenzhen, China
[2]Department of Computer Science & IT, University of Sargodha, Pakistan
[3]School of Science, Harbin Institute of Technology (Shenzhen), Shenzhen, China
[4]Department of Computer Science and Engineering, Hebei University of Technology, Tianjin, China
{msaqibnawaz, philfv}@hit.edu.cn, zohaib.nawaz@uos.edu.pk, chenguoting@hit.edu.cn, wuc567@163.com

Abstract. Application Programming Interface (API) calls in windows operating system (OS) is an attractive feature for malware analysis and detection as they can properly reflect the actions of portable executable (PE) files. In this paper, we provide an approach based on sequential pattern mining (SPM) for the analysis of malware behavior during executions. A dataset that contains sequences of API calls made by different malware on Windows OS is first abstracted into a suitable format (sequences of integers). SPM algorithms are then used on the corpus to find frequent API calls and their patterns. Moreover, sequential rules between API calls patterns as well as maximal and closed frequent API calls are discovered. Obtained preliminary results suggest that discovered frequent patterns of API calls and sequential rules between them can be used in the development of malware detection and classification techniques.

Keywords: Malware analysis, Sequential pattern mining, API calls, Frequent patterns, Sequential rules.

1 Introduction

The name malware (malicious software) is used collectively for a number of software that is designed primarily to access restricted data, block data to get ransom and damage or destruct computers and mobile devices without owners' knowledge or permission [7,32]. Since their emergence in the late 1980s, malware pose a serious threat to computing systems and networks. Although computing devises have become more secure with each new Operating System (OS) update, attackers can still bypass these security components by using different methods. Due to COVID-19, cybercrime is up 600%, and in the last ten years, there has

been an 87% increase in malware infections. Moreover, malware attacks in 2019 have costed the average US company an average of $2.4 million per year[1].

As malware can cause enormous loss and adverse effects, its analysis and detection is an important research topic in cybersecurity. Anti-malware software [16] that provides protection against malware mainly use signature-based method for malware detection. During the scanning process, these tools search for unique signatures (short and unique strings set) that are already extracted from known malicious files. An executable file is identified as a malicious code if its signature matches with the list of available signatures. This approach is fast in identifying known malware. However, extracting signature is a tedious and time consuming activity and requires funds and expertise. Moreover, signature-based method looks for already known malicious patterns, thus it is unreliable and ineffective against the new malicious codes [7]. Nowadays, new malware samples are created at a high speed. For example, recent McAfee report[2] says that approximately 77 M new malicious activities were detected in Q1 2021.

To overcome aforementioned drawbacks, data mining and machine learning techniques are now used for the analysis of malware and in the development of efficient malware classification and detection techniques. In this study, we focus on Windows based-malware as this OS is by far the most popular desktop OS with a market share of about 72.98%[3]. On Windows OS, a malware needs to use some of the OS's services. The entire set of requests made by malware to get these services through application programming interface (API) calls creates malicious behaviour. We believe that sequential pattern mining (SPM) [13] is well suited to analyse sequences of API calls made by malware. SPM, a special case of structured data mining, is used to discover meaningful and hidden knowledge in large sequential datasets. SPM has been used extensively in the past in various applications such as bioinformatics [1, 20], market basket analysis [23], text analysis [19, 25], energy reduction in smarthomes [29], webpage click-stream analysis [10] and proof sequences [21, 22].

In this study, the focus is on metamorphic malware. An SPM-based approach is presented for the analysis of API calls sequences made by malware that were run in an isolated secured environment. The basic idea of the approach is to first convert the API calls sequences into a corpus that is suitable for learning, where each API call is converted into an integer. SPM techniques are then used on the corpus to find frequent API calls and their patterns. Moreover, sequential rules between API calls patterns as well as maximal and closed frequent API calls are discovered. We believe that obtained results can be used in the development of malware detection and classification techniques, where frequent patterns of API calls and the sequential rules between them can be used for the classification and detection process.

The remainder of this paper is organized as follows. Section 2 describes the related work on using data mining and machine learning in the analysis of mal-

[1] legaljobs.io/blog/malware-statistics
[2] http://mcafee.com/enterprise/en-us/lp/threats-reports/apr-2021.html
[3] http://gs.statcounter.com/os-market-share/desktop/worldwide

ware API calls. The details of the dataset that we used in this work is provided in Section 3, followed by the proposed SPM-based approach that is used for the analysis of API calls in Section 4. Section 5 presents and discusses the obtained results. Finally, Section 6 concludes the paper with some future research directions.

2 Related Work

API was used first as a feature of program in [17] where a method was developed for anomaly intrusion detection based on sequences of system calls. API calls were also used in [33] for the development of an intelligence malware detection system (IMDS). IMDS performance was much better than signature-based and other data mining-based methods in terms of detection rate and classification accuracy. A dynamic malware detection system was developed in [3] that was also based on API calls of malware. The work [31] used the call grams and odds ratio selection to discover the distinct API sequences. These sequences were then used as inputs to the classifiers to categorize malware and benign samples. API call sequences were transformed into byte-based sequential data in [26] that was used later in the detection process. The malware detection approach in [30] used text mining and topic modeling for feature extraction and selection based on the API call sequences.

To capture the common API sequence among different malware types, the longest common sequence (LCS) algorithm was used in [18] and multiple sequence alignment (MSA) algorithms were used in [6] to find common API calls sequences and their classification into different malware families. Frequent API names and their arguments were used in [27] to represent the behaviour of a malware. The Apriori algorithm [2] was used to mine frequent itemsets composed of API names and/or API arguments. However, the Apriori algorithm does not take into account the sequential ordering of events, such as API calls. Thus, it fails to discover important patterns and also ignore the sequential relationship between events. Moreover, the malware detection method in [24] used one SPM algorithm to discover frequent API call patterns of malware. Three classifiers were then used on the patterns for the classification of malware samples. Authors claim that they discovered discriminative frequent API call patterns. However, the algorithm that they used was CM-SPAM which is generally used to find common sequential patterns, not discriminative patterns. Moreover, the dataset is not discussed in detail and no link is provided to download the dataset.

Compared to previous works, we used a dataset for API calls of metamorphic malware. Moreover, not one but various SPM algorithms are used on the dataset to discover not only frequent API calls and their patterns, but also the sequential rules between discovered patterns as well as closed and maximal sequential patterns of API calls. Using more than one SPM algorithm also allowed us to check their effectiveness on the dataset of API calls sequence of malware that we discuss in the next section.

3 Dataset

The benchmark dataset [4,5] contains Windows API calls of various metamorphic malware such as WannaCry and Zeus. An application running on Windows uses APIs to utilize a function that the OS provides. The use of an OS functions is called an API call. During execution, an application can makes many API calls. For example, to create a file, an application calls the *CreateFileA* API. Similarly, the API call *DeviceIoControl* is used by an application to perform direct input and output operations on, or retrieve information about, a specified device driver. Thus, API calls are generally used in dynamic malware analysis. A metamorphic malware can change its structure continuously by making changes in their source code and signature patterns with each iteration.

In the dataset [4,5], approximately 20,000 malware were collected from GitHub and their MD5 (Message-Digest algorithm 5) hash values were obtained. Cuckoo Sandbox Environment was then used to check the behavior of each malware separately in an isolated environment. The final data in the dataset contains all Windows API call requests made by the malware on Windows 7 OS. The API calls sequences are further filtered and those rows in the sequences were discarded that did not contain at least 10 different API calls. The hash values of malware were searched with the VirusTotal service and the analysis results from the VirusTotal service was stored in a database. Each malware's families was identified by using each analysis result that was obtained with VirusTotal.

The final dataset contains 8 different leading malware families, which are: *Trojan, Backdoor, Downloader, Worms, Spyware Adware, Dropper* and *Virus*. Table 1 shows the number of malware belonging to malware families in the whole dataset.

Table 1: Malware distribution according to their families

Malware Family	**Samples**	**Considered**
Spyware	832	21
Downloader	1001	91
Trojan	1001	585
Worms	1001	92
Adware	379	9
Dropper	891	19
Virus	1001	36
Backdoor	1001	147
Total	7101	1000

The whole dataset contains 7,101 sequences in total. In this work, we considered the first 1000 sequences from the dataset. Considered sequences belonged to different malware types and the detail of the sequences belonging to which malware type is provided in Table 1. In the first 1000 sequences, approximately 60% of sequences belonged to the *Trojan* family.

4 Analyzing API Calls with SPM

The structure of the SPM-based approach for the analysis of API calls made by different malware is shown in Figure 1. It consists of two main parts:

1. *Converting different malware API calls dataset into a suitable format for SPM*: The sequences of API calls made by malware are transformed in a suitable abstraction, so that no meaningful information from the sequences are left out. For this, we use the "*API calls sequences to integers*" abstraction, where each API call type is converted into a distinct positive integer. Such abstraction allows wide diversity and makes the approach more general in nature.
2. *Learning through SPM*: SPM algorithms are used on the corpus to discover the common API calls and their frequent sequences, sequential relationships of frequent API calls and their patterns with each other and frequent closed and maximal frequent API calls.

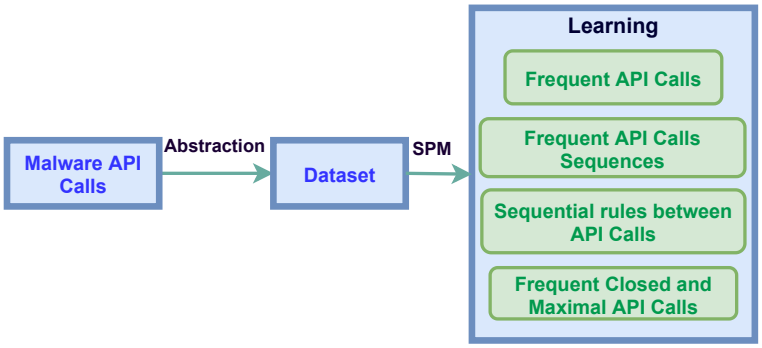

Fig. 1: Proposed SPM-based approach

In the following we present some concepts related to sequences in the context of this work for better understanding.

Let $AC = \{ac_1, ac_2, ..., ac_m\}$ represent the set of API calls. An *API calls set* ACS is a set of API calls such that $ACS \subseteq AC$. $|ACS|$ denotes the set cardinality. An ACS has a length k (called k-ACS) if it contains k API calls, i.e., $|ACS| = k$. For example, consider the set of API calls $PS = \{$*LdrLoadDll, NtOpenKey, CopyFileA, FindResourceA, GetFileSize, Exception*$\}$. The following set $\{$*LdrLoadDll*, *NtOpenKey*, *Exception*$\}$ is an API calls set that contains three API calls. An order relation on API calls is defined by assuming that there exists a total order on API calls $\prec$ such as the lexicographical order.

An API calls sequence is a list of API calls sets $S = \langle ACS_1, ACS_2, ..., ACS_n \rangle$, such that $ACS_i \subseteq ACS$ $(1 \leq i \leq n)$. An *API calls corpus* ACC is a list of API calls sequences $ACC = \langle S_1, S_2, ..., S_n \rangle$, where each sequence has an

identifier (ID). For example, Table 2 shows an *ACC* that has four API calls sequences with IDs 1, 2, 3 and 4.

Table 2: A sample of an API calls corpus

ID	API calls sequence
1	$\langle\{$*LdrLoadDll, RegOpenKeyExA, NtOpenKey, NtQueryValueKey*$\}\rangle$
2	$\langle\{$*NtClose, DeviceIoControl, NtClose, NtClose, NtReadFile, WSAStartup*$\}\rangle$
3	$\langle\{$*NtCreateFile, GetFileSize, NtClose, NtReadFile, DrawTextExA, NtDelayExecution, NtClose, GetKeyState*$\}\rangle$
4	$\langle\{$*FindWindowExW, FindFirstFileExA, OpenKey, NtClose, NtClose, NtCreateFile, LdrGetProcedureAddress, GetAsyncKeyState, GetKeyState, DeviceIoControl, NtClose, NtDelayExecution*$\}\rangle$

The final step is to convert the API calls into sequences of integers to bring the dataset in a more suitable format for SPM techniques. In the final corpus, each line represents an API calls sequence for a malware. Each API call in the sequence is replaced by a positive integer. For example, the API call *NtClose* is replaced with 6. Moreover, API calls are separated with a single space followed by a negative integer -1. Negative integer -2 appears at the end of each line that shows the end of a sequence. The four API calls sequences in Table 2 are converted into the integer sequences shown in Table 3.

Table 3: Conversion of API calls sequences into integer sequences

ID	API call sequence
1	1 -1 3 -1 4 -1 5 -1 -2
2	6 -1 12 -1 6 -1 6 -1 32 -1 23 -1 -2
3	7 -1 11 -1 6 -1 32 -1 15 -1 18 -1 6 -1 22 -1 -2
4	19 -1 22 -1 14 -1 6 -1 6 -1 7 -1 19 -1 31 -1 22 -1 12 -1 6 -1 18 -1 -2

An API calls sequence $S_\alpha = \langle\alpha_1, \alpha_2, ..., \alpha_n\rangle$ is present or contained in another API calls sequence $S_\beta = \langle\beta_1, \beta_1, ..., \beta_m\rangle$ iff there exist integers $1 \leq i_1 < i_2 < ... < i_n \leq m$, such that $\alpha_1 \subseteq \beta_{i1}, \alpha_2 \subseteq \beta_{i2}, ..., \alpha_n \subseteq \beta_{im}$ (denoted as $S_\alpha \sqsubseteq S_\beta$). If S_α is present in S_β, then S_α is a *subsequence* of S_β. In SPM, various measures are used to investigate the importance and interestingness of a subsequence. The *support* measure is used by most SPM techniques. The *support* of S_α in *ACC* is the total number of sequences (S) that contain S_α, and is represented by $sup(S_\alpha)$:

$$sup(S_\alpha) = |\{S|S_\alpha \sqsubseteq S \wedge S \in ACC\}|$$

SPM is an enumeration problem that aims to find all the *frequent subsequences* in a sequential dataset. S is a *frequent sequence* (also called *sequential*

pattern) iff $sup(S) \geq minsup$, where *minsup* (minimum support) is the threshold being determined by the user. A sequence containing n items (API calls in this work) in a corpus can have up to $2^n - 1$ distinct subsequences. This makes the naive approach to calculate the support of all possible subsequences infeasible for most corpora. Several efficient algorithms have been developed in recent years that do not explore all the search space for all possible subsequences.

All SPM algorithms investigate the patterns search space with two operations: *s-extensions* and *i-extensions*. A sequence $S_\alpha = \langle \alpha_1, \alpha_2, ..., \alpha_n \rangle$ is a *prefix* of another sequence $S_\beta = \langle \beta_1, \beta_1, ..., \beta_m \rangle$, if $n < m$, $\alpha_1 = \beta_1$, $\alpha_2 = \beta_2$, ..., $\alpha_{n-1} = \beta_{n-1}$, where α_n is equal to the first $|\alpha_n|$ items of β_n according to the $\prec$ order. Note that SPM algorithms follow a specific order $\prec$ so that the same potential patterns are not considered twice and the choice of the order $\prec$ does not affect the final result produced by SPM algorithms. A sequence S_β is an *s-extension* of a sequence S_α for an item x if $S_\beta = \langle \alpha_1, \alpha_2, ..., \alpha_n, \{x\} \rangle$. Similarly, for an item x, a sequence S_γ is an *i-extension* of S_α if $S_\gamma = \langle \alpha_1, \alpha_2, ..., \alpha_n \cup \{x\} \rangle$. SPM algorithms either employ breadth-first search or depth-first search algorithms. In the following, a brief description of state-of-the-art SPM algorithms is presented.

In frequent itemset mining (FIM), the first and most famous algorithm is the Apriori algorithm [2]. It can find frequent itemsets in large databases. It proceeds by discovering common items that can be extended to larger itemsets that appear sufficiently often. Itemsets (ACS in this work) extracted by Apriori can also be used to identify association rules (relationships) between items. Over the years, several fast and memory efficient FIM algorithms have been proposed.

The TKS (Top-k Sequential) algorithm finds the top-k sequential patterns in a corpus, where k is set by the user and it represents the number of sequential patterns to be discovered by the algorithm. TKS employs the basic candidate generation procedure of SPAM and vertical database representation. With vertical representation, support for patterns can be calculated without performing costly database scans. This makes vertical algorithms to perform better on dense or long sequences. TKS also uses several strategies for search space pruning and depends on the PMAP (Precedence Map) data structure to avoid costly operations of bit vector intersection. Another SPM algorithm is the CM-SPAM algorithm that examines the whole search space to discover frequent sequential patterns in the corpus. The CMAP (Co-occurrence MAP) data structure is used in CM-SPAM to store co-occurrence of item information. A generic pruning mechanism that is based on CMAP is used for pruning the search space with vertical database representation, to efficiently discover sequential patterns. More detail on TKS and CM-SPAM can be found in [9] and [8] respectively.

However, one main drawback of aforementioned algorithms is that they may find too many sequential patterns for users. The complexity and generated patterns can be reduced by mining closed and maximal sequential patterns. A sequential pattern s_a is said to be *closed* if there is no other sequential pattern s_b, such that s_b is a superpattern of s_a, $s_a \sqsubseteq s_b$, and their supports are equal. A sequential pattern s_a is said to be *maximal* if there is no other sequential pattern s_b, such that s_b is a superpattern of s_a, $s_a \sqsubseteq s_b$. The problem of mining

closed (maximal) sequential patterns is to discover the set of closed (maximal) sequential patterns. CloFAST (Closed FAST sequence mining algorithm based on sparse id-lists) [15] and VMSP (Vertical mining of Maximal Sequential Patterns) [14] are the two algorithms used for finding closed and maximal sequential patterns, respectively, in large databases.

Sequential patterns that appear frequently in a corpus with low confidence are worthless for decision making or prediction. Sequential rules discover patterns by considering not only their support but also their confidence. A sequential rule $X \rightarrow Y$ is a relationship between two ACSs $X, Y \subseteq AC$, such that $X \cap Y = \emptyset$ and $X, Y \neq \emptyset$. The rule $r : X \rightarrow Y$ means that if items of X occur in a sequence, items of Y will occur afterward in the same sequence. X is contained in S_α (written as $X \sqsubseteq S_\alpha$) iff $X \subseteq \bigcup_{i=1}^{n} \alpha_i$. A rule $r : X \rightarrow Y$ is contained in S_α ($r \sqsubseteq S_\alpha$) iff there exists an integer k such that $1 \leq k < n$, $X \subseteq \bigcup_{i=1}^{k} \alpha_i$ and $Y \subseteq \cup_{i=k+1}^{n} \alpha_i$. The confidence of r in ACC is defined as:

$$conf_{ACC}(r) = \frac{|\{S | r \sqsubseteq S \wedge S \in ACC\}|}{|\{S | X \sqsubseteq S \wedge S \in ACC\}|}$$

The support of r in ACC is defined as:

$$sup_{ACC}(r) = \frac{|\{S | r \sqsubseteq S \wedge S \in ACC\}|}{|ACC|}$$

A rule r is a *frequent sequential rule* iff $sup_{ACC}(r) \geq minsup$ and r is a *valid sequential rule* iff it is frequent and $conf_{ACC}(r) \geq minconf$, where the thresholds $minsup, minconf \in [0, 1]$ are set by the user. Mining sequential rules in a corpus deals with finding all the valid sequential rules. For this, the ERMiner (Equivalence class based sequential Rule Miner) algorithm [11] is used. It relies on a vertical database representation and the search space of rules is explored using the equivalence classes of rules with the same antecedent and consequent. It employs two operations (left and right merges) to explore the search space of frequent sequential rules, where the search space is pruned with the Sparse Count Matrix (SCM) technique, which makes ERMiner more efficient than other sequential rule finding algorithms.

5 Results and Discussion

All the following experiments are performed on an HP laptop with a fifth generation Core i5 processor and 4 GB RAM. SPMF data mining library, developed in JAVA, is used to analyze the API calls corpus. It is an open-source and cross-platform framework that is specialized in pattern mining tasks. SPMF offers implementations for more than 180 data mining algorithms. More detail on SPMF can be found in [12].

We first used SPMF to convert the API calls sequences into integer sequences that is suitable for SPM algorithms. The first 1000 sequences contain 254 distinct API calls in total and on average, each sequence contains approximately

11,502 API calls. Results obtained by applying SPM algorithms on the corpus are discussed next. Interested readers can find the converted dataset used in this work at: *github.com/saqibdola/MBAwSPM*.

The Apriori algorithm was first applied on the corpus to find the frequently occurring API calls. Apriori takes a corpus and a *minsup* threshold as input and outputs the frequent API calls. The sets extracted by Apriori are listed in Table 4. For high *minsup* values, Apriori generated less frequent patterns. By decreasing *minsup* to 10%, Apriori generated 274 patterns. The top five freuquent API calls in 1000 sequences were *GetAsyncKeyState*, *GetKeyState*, *DeviceIoControl*, *NtClose* and *NtDelayExecution* with 2,343,660, 1,103,492, 800,088, 680,206 and 573,625 occurrences, respectively.

Table 4: Frequent API Calls extracted by Apriori

Frequent API Calls Count	**Min. Sup**
117	100%
118	90%
120	80%
123	70%
127	60%
135	50%
141	40%
155	30%
170	20%
274	10%

The frequent API calls sets obtained with Apriori are uninteresting in the sense that they are unordered as they do not follow any specific order. Moreover, Apriori does not ensure that API calls from a API calls set appear contiguously in the sequence. Apriori fails to discover important sequential patterns and also ignore the sequential relationship between API calls. Next, we present the results from the application of SPM algorithms that overcome the drawbacks of Apriori. Thus they reveal more meaningful patterns and information.

The TKS algorithm is applied on the corpus to find top-k sequential patterns of API calls. TKS takes a corpus and a user specified parameter k as input and returns the top-k most frequent patterns as output. Some API calls frequent patterns discovered by the TKS algorithm with varying length are shown in Table 5. Note that the column **Sup** indicates the occurrence count of each pattern in the corpus. Table 5 provides some useful information related to frequent occurrences of API calls and patterns in the sequences. The second last column shows that the same API call *NtClose* as a sequence of six occurred 920 times in the corpus. Similarly, the API call in the last column *LdrGetProcedureAddress* occurred as a sequence of ten 880 times in the corpus.

Unlike TKS, the CM-SPAM algorithm offers the *minsup* threshold. Table 6 lists some of the most useful frequent API calls patterns in the corpus which are extracted with the CM-SPAM algorithm. The first four API calls patterns appear

Table 5: Extracted API calls patterns with TKS algorithm

Pattern	**Sup**
NtClose, NtQueryValueKey, NtClose	919
LdrLoadDll, LdrGetProcedureAddress, NtClose	904
NtClose, NtOpenKey NtQueryValueKey, NtClose	915
NtClose, NtOpenKey, NtClose, NtClose, NtClose	916
NtClose, NtOpenKey, NtClose, NtClose, NtClose, NtClose	916
NtAllocateVirtualMemory, NtClose, NtClose	882
6 × *NtClose*	920
10 × *LdrGetProcedureAddress*	880

in at least 90% of the sequences in the corpus. The next four patterns appear in at least 80% of the sequences. Discovered patterns with the CM-SPAM algorithm are almost similar to the results obtained with the TKS algorithm. It was found that API call *NtClose* appeared in almost every frequent patterns discovered using TKS and CM-SPAM. Moreover, in 254 total API calls, approximately 15 API calls appeared mostly in frequent patterns.

Table 6: Frequent API calls patterns extracted with CM-SPAM

Pattern	**Min. Sup**	**Sup**
NtClose, NtQueryvalueKey, NtClose	0.9	919
NtClose, NtQueryValueKey, NtqueryValueKey, NtClose	0.9	915
NtOpenKey, NtClose, NtClose, NtClose	0.9	934
7 × *NtClose*	0.9	900
NtCreateFile, NtClose, NtClose	0.8	803
LdrGetDllHandle, LdrGetDllHandle, NtClose, NtClose	0.8	841
NtFreeVirtualMemory, NtClose, NtClose	0.8	803
12 × *NtClose*	0.8	816

Table 7 shows the relationships between frequent API calls that are discovered through sequential rule mining with the ERMiner algorithm. The confidence ($misconf$) threshold is set to 80%, which means that rules have a confidence of at least 80% (a rule $X \rightarrow Y$ has a confidence of 80% if the set of API calls in X is followed by the set of API calls in Y at least 80% of the times when X appears in a sequence). For example, the first rule in Table 7 indicates that 94.5% of the time, the API call *LdrLoadDll* is followed after *LdrGetProcedureAddress* and this rule has occurred 898 times in the corpus. Similarly, the second rule is the opposite of the first rule: 98.9% of the time, the API call *LdrGetProcedureAddress* is followed after *LdrLoadDll* 917 times in the corpus. With ERminer, we found some interesting relationships and dependencies between frequent API calls.

Some of the discovered closed and maximal sequential API calls patterns are listed in Table 8. The closed sequential patterns provide a lossless representation of all sequential patterns and they represent the largest subsequences common

Table 7: Discovered sequential rules

Rule	Sup	Conf
LdrGetProcedureAddress → *LLdrLoadDll*	898	0.945
LdrLoadDll → *LdrGetProcedureAddress*	917	0.989
NtClose → *LdrLoadDll*	859	0.86
LdrLoadDll → *NtClose*	913	0.98
LdrLoadDll, NtAllocateVirtualMemory → *LdrGetProcedureAddress*	859	0.97
LdrLoadDll, LdrGetProcedureAddress → *NtAllocateVirtualMemory*	841	0.91
LdrLoadDll, NtAllocateVirtualMemory → *NtClose*	871	0.98
LdrLoadDll, NtClose→ *NtAllocateVirtualMemory*	810	0.88
LdrLoadDll, LdrGetProcedureAddress, NtQueryValueKey → *NtClose*	855	0.99
NtClose →*LdrLoadDll, LdrGetProcedureAddress, NtQueryValueKey*	808	0.81
LdrGetProcedureAddress, NtClose → *NtOpenKey, NtQueryValueKey*	834	0.88
LdrGetProcedureAddress, NtOpenKey → *NtQueryValueKey, NtClose*	840	0.93
LdrLoadDll, NtOpenKey, NtQueryValueKey, LdrGetDllHandle → *NtClose*	808	0.991

to sets of sequences. For example, if each sequence represents the behavior of a malware, the closed patterns represent the largest patterns common to groups of malware sequences. Whereas the maximal API calls patterns are not loseless and is always not larger than the set of closed sequential patterns and all sequential patterns. Note that in maximal sequential API patterns, the *max gap* is set to 1. This means that no gap is allowed and each consecutive API call of a pattern appears consecutively in a sequence.

Table 8: Closed (C) and Maximal (M) frequent sequential API patterns

Closed	Sup
NtAllocateVirtualMemory, NtAllocateVirtualMemory	729
NtClose, NtOpenKey, NtQueryValueKey	755
NtFreeVirtualMemory, NtClose, NtClose	803
NtOpenKey, NtQueryValueKey, NtClose, NtClose	700
LdrGetDllHandle, 4 × *LdrGetProcedureAddress*	702
16 × *LdrGetProcedureAddress*	702

Maximal	Sup
LdrLoadDll, LdrGetDllHandle	842
LdrLoadDll, LdrLoadDll, LdrGetDllHandle	814
LdrGetDllHandle, LdrGetProcedureAddress, NtOpenKey	806
LdrLoadDll, NtClose, NtOpenKey	817
3 × *LdrGetProcedureAddress, LdrLoadDll, LdrGetDllHandle*	807
36 × *LdrGetProcedureAddress*	801

During execution, we found that all the algorithms worked efficiently on the corpus and results obtained indicated that the total number of API calls in each

sequence (abstraction simplicity) has a direct correlation on the efficiency of SPM algorithms.

6 Conclusion

In this paper, sequential pattern mining (SPM) algorithms were used to analyse malware behavior, which is an important research topic in cybersecurity. A dataset that contain API calls sequences of various metamorphic malware was first converted into a suitable format. Various SPM algorithms were then used on the abstracted dataset to find frequent API calls, frequent API calls sequences, the sequential relationships between frequent patterns as well as closed and maximal frequent API calls patterns. All the algorithms performed efficiently on the dataset and some interesting patterns were found with SPM. There are several directions for future work, some of which are:

- Applying SPM algorithms on the whole dataset and also on each malware types. This will allow us to discover frequent patterns of each malware and also investigate their behavior separately.
- Predicting the next API calls in a sequence using prediction models offered by SPMF.
- Using emerging patterns mining or contrast set mining techniques [28] on the API calls of various malware to discover emerging (or contrasting) trends that show a clear and useful difference (or contrast) between various malware behavior.
- Taking the frequent patterns of API calls discovered by TKS, CM-SPAM and sequential rules between frequent API calls discovered by ERMiner as features for the development of malware samples classification and detection.

References

1. M. Abouelhoda and M. Ghanem. String mining in bioinformatics. In *Scientific Data Mining and Knowledge Discovery*, pages 207–247. 2010.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of VLDB*, pages 487–499, 1994.
3. M. Ahmadi, A. Sami, H. Rahimi, and B. Yadegari. Malware detection by behavioural sequential patterns. *Computer Fraud and Security*, 2013(8):11–19, 2013.
4. F. Ö. Çatak and A. F. Yazi. A benchmark API call dataset for windows PE malware classification. *CoRR*, abs/1905.01999, 2019.
5. F. Ö. Çatak, A. F. Yazi, O. Elezaj, and J. Ahmed. Deep learning based sequential model for malware analysis using Windows exe API calls. *PeerJ Computer Science*, 6:e285, 2020.
6. I. K. Cho and E. G. Im. Extracting representative API patterns of malware families using multiple sequence alignments. In *Proceedings of RACS*, pages 308–313, 2015.
7. Y. Fan, Y. Ye, and L. Chen. Malicious sequential pattern mining for automatic malware detection. *Expert Systems with Applications*, 52:16–25, 2016.

8. P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas. Fast vertical mining of sequential patterns using co-occurrence information. In *Proceedings of PAKDD*, pages 40–52, 2014.
9. P. Fournier-Viger, A. Gomariz, T. Gueniche, E. Mwamikazi, and R. Thomas. TKS: Efficient mining of top-k sequential patterns. In *Proceedings of ADMA*, pages 109–120, 2013.
10. P. Fournier-Viger, T. Gueniche, and V. S. Tseng. Using partially-ordered sequential rules to generate more accurate sequence prediction. In *Proceedings of ADMA*, pages 431–442, 2012.
11. P. Fournier-Viger, T. Gueniche, S. Zida, and V. S. Tseng. ERMiner: Sequential rule mining using equivalence classes. In *Proceedings of IDA*, pages 108–119, 2014.
12. P. Fournier-Viger, J. C. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam. The SPMF open-source data mining library version 2. In *Proceedings of the ECML/PKDD*, pages 36–40, 2016.
13. P. Fournier-Viger, J. C. W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.
14. P. Fournier-Viger, C. Wu, A. Gomariz, and V. S. Tseng. VMSP: Efficient vertical mining of maximal sequential patterns. In *Proceedings of CCAI*, pages 83–94, 2014.
15. F. Fumarola, P. F. Lanotte, M. Ceci, and D. Malerba. CloFAST: Closed sequential pattern mining using sparse and vertical id-lists. *Knowledge and Information Systems*, 48(2):429–463, 2016.
16. K. Griffin, S. Schneider, X. Hu, and T.-c. Chiueh. Automatic generation of string signatures for malware detection. In *Proceesings of RAID*, pages 101–120, 2009.
17. S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.
18. Y. Ki, E. Kim, and H. K. Kim. A novel approach to detect malware based on API call sequence analysis. *International Journal of Distributed Sensor Networks*, 11:659101:1–659101:9, 2015.
19. R. U. Mustafa, M. S. Nawaz, J. Ferzund, M. I. U. Lali, B. Shahzad, and P. Fournier-Viger. Early detection of controversial urdu speeches from social media. *Data Science and Pattern Recognition*, 1(2):26–42, 2017.
20. M. S. Nawaz, P. Fournier-Viger, A. Shojaee, and H. Fujita. Using artificial intelligence techniques for COVID-19 genome analysis. *Applied Intelligence*, 51(5):3086–3103, 2021.
21. M. S. Nawaz, P. Fournier-Viger, and J. Zhang. Proof learning in PVS with utility pattern mining. *IEEE Access*, 8:119806–119818, 2020.
22. M. S. Nawaz, M. Sun, and P. Fournier-Viger. Proof guidance in PVS with sequential pattern mining. In *Proceedings of FSEN*, pages 45–60, 2019.
23. L. Ni, W. Luo, N. Lu, and W. Zhu. Mining the local dependency itemset in a products network. *ACM Transactions on Management Information Systems*, 11(1):3:1–3:31, 2020.
24. A. Pektas, E. N. Pektas, and T. Acarman. Mining patterns of sequential malicious APIs to detect malware. *International Journal of Network Security & Its Applications*, 10(4):1–9, 2018.
25. Y. J. M. Pokou, P. Fournier-Viger, and C. Moghrabi. Authorship attribution using small sets of frequent part-of-speech skip-grams. In *Proceedings of FLAIRS*, pages 86–91, 2016.
26. Y. Qiao, Y. Yang, J. He, C. Tang, and Z. Liu. CBM: Free, automatic malware analysis framework using API call sequences. In *Knowledge Engineering and Management*, pages 225–236, 2014.

27. Y. Qiao, Y. Yang, L. Ji, and J. He. Analyzing malware by abstracting the frequent itemsets in API call sequences. In *Proceedings of TrustCom*, pages 265–270, 2013.
28. V. S and J. M. Luna. *Supervised Descriptive Pattern Mining*. Springer, 2018.
29. D. Schweizer, M. Zehnder, H. Wache, H. F. Witschel, D. Zanatta, and M. Rodriguez. Using consumer behavior data to reduce energy consumption in smart homes: Applying machine learning to save energy without lowering comfort of inhabitants. In *Proceedings of ICMLA*, pages 1123–1129, 2015.
30. G. G. Sundarkumar, V. Ravi, I. Nwogu, and V. Govindaraju. Malware detection via API calls, topic models and machine learning. In *Proceedings of CASE*, pages 1212–1217, 2015.
31. D. Uppal, R. Sinha, V. Mehra, and V. Jain. Malware detection and classification based on extraction of API sequences. In *Proceedings of ICACCI*, pages 2337–2342, 2014.
32. Y. Ye, T. Li, D. A. Adjeroh, and S. S. Iyengar. A survey on malware detection using data mining techniques. *ACM Computing Surveys*, 50(3):41:1–41:40, 2017.
33. Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang. An intelligent PE-malware detection system based on association mining. *Journal of Computer Virology*, 4(4):323–334, 2008.