Malware Characterization Using Windows API Call Sequences

Sanchit Gupta[1,*], Harshit Sharma[2] and Sarvjeet Kaur[1]

[1]*SAG, DRDO, Delhi, India*
[2]*NIIT University, Neemrana, India*
E-mail: sanchitgupta@sag.drdo.in; sarvjeet@sag.drdo.in; harshit.sharma@st.niituniversity.in
[*]*Corresponding Author*

Received 05 April 2017; Accepted 29 June 2018;
Publication 27 July 2018

Abstract

In this research we have used Windows API (Win-API) call sequences to capture the behaviour of malicious applications. Detours library by Microsoft has been used to hook the Win-APIs call sequences. To have a higher level of abstraction, related Win-APIs have been mapped to a single category. A total set of 534 important Win-APIs have been hooked and mapped to 26 categories (A...Z). Behaviour of any malicious application is captured through sequence of these 26 categories of APIs. In our study, five classes of malware have been analyzed: Worm, Trojan-Downloader, Trojan-Spy, Trojan-Dropper and Backdoor. 400 samples for each of these classes have been taken for experimentation. So a total of 2000 samples were taken as training set and their API call sequences were analyzed. For testing, 120 samples were taken for each class. Fuzzy hashing algorithm ssdeep was applied to generate fuzzy hash based signature. These signatures were matched to quantify the API call sequence homologies between test samples and training samples. Encouraging results have been obtained in classification of these samples to the above mentioned 5 categories. Further, N-gram analysis has also been done to extract different API call sequence patterns specific to each of the 5 categories of malware.

Keywords: Win-API, API hooking, malware, fuzzy hashing.

1 Introduction

In today's world everyone is connected and uses internet for most of the things. This not only creates dependency on the internet but also increases possibility of exploitation via it. Besides computers, smartphones are also a great source of connectivity. Managing ever-evolving malware related to these devices is critical for proper functioning and security. Despite the use of anti-virus software, new malware and their variants are spreading continuously. Worms, Backdoors and Trojans are growing at tremendous rate thus affecting the secrecy, integrity and functionality of the systems. Thus the researchers and anti-malware vendors are always working in the area of developing new solutions to counter the effect of malware.

Various approaches like Static analysis and Dynamic analysis have been proposed for activities related to malware analysis. In Static analysis the binary code is analyzed without executing it, whereas in Dynamic analysis the code is executed and its behaviour is monitored. The advantage in dynamic analysis is that it even works for sophisticated obfuscated binaries where static analysis is quite challenging and time-consuming. However static features like Opcode n-gram, Byte code n-gram have been used as features for Malware detection systems [1–4].

New malware can easily evade traditional hash-based signature detection by just introducing slight modification in the code or applying obfuscation techniques. But signatures based on dynamic analysis provide better detection rate as they capture the behaviour of the malware which remains unaltered even after obfuscation. Further to categorize the malware in different classes, behaviour specific to particular class needs to be identified.

The main advantage in dynamic analysis is that the run-time behaviour of the executable is difficult to obfuscate. Also, the dynamic malware analysis can be easily automated enabling analysis at large scale possible. But the disadvantage of dynamic analysis is that it captures only one execution trace of the whole program. Also the program must be run in secure run-time environment to evade the danger of getting infection while doing analysis. Both of these limitations can be addressed by using good test vectors for maximum code coverage and setting safe virtual environment. Egele et al. [5] given an extensive survey of dynamic malware analysis techniques. We have used dynamic analysis technique to analyze different class samples, where-in API call sequences are extracted by running the samples.

Using API-calls for dynamic malware analysis is not a new concept as many techniques have been proposed in the literature. Santos et al. [6] proposes a malware detector based upon frequency of occurrence of operational code and API-calls. Ye et al. [7] proposed malware detection system based on interpretable string analysis and uses SVM with bagging for classification purpose. Zolkipli and Jantan [8] presented malware behaviour identification using run time analysis and resource monitoring and malware classification using artificial intelligence technique. Islam et al. [9] used static parameters namely string information, function length frequency and dynamic parameters namely API function name and function parameters to classify between malware and clean files. Gandotra et al. [10] gave extensive survey of various researches related to malware classification. Ranveer and Hirai [11] categorized various features used in the malware detection systems. They have compared features of static, dynamic and hybrid type. Youngjoon et al. [12] used API call sequences as features and they claimed to get accuracy of 0.998 in classification between benign and malware samples. They have used DNA sequence alignment algorithm for detection of malware samples.

Above mentioned research mainly is in the area of classification between malware and benign samples. Nothing much has been done in regard of subclassification between various families of malwares. Park et al. [13] classified various variants of worms based on system call graph matching. They used maximal common subgraph as a feature to find similarities in worms. But their model is not able to provide higher classification accuracy. Nari et al. [14] presented a framework for malware classification into their respective families based on only the network behaviour. They have used network flow and their dependencies to build behavioural profile. Families considered for classification were variants of trojans, backdoors and worms. Their framework would not classify malwares with no network signatures.

The techniques on analysis of API-calls in conjunction with permissions and system call behavior has also been used for classification of Android based malwares. Here also dynamic analysis is performed to generate behavior patterns from malicious APKs [15, 16]. Effectiveness of these results in Android environment motivated us to perform such experiments on Windows based malwares.

In our study, five classes of malware have been analyzed: Worm, Trojan-Downloader, Trojan-Spy, Trojan-Dropper and Backdoor. We took the main classes of windows malware and observed their behaviours related to files, registry, network, services etc. by observing total 534 API calls related to each category. The main contribution in this paper is developing a technique for malware classification and further extracting signatures for all these five classes based on API call sequences.

2 Methodology

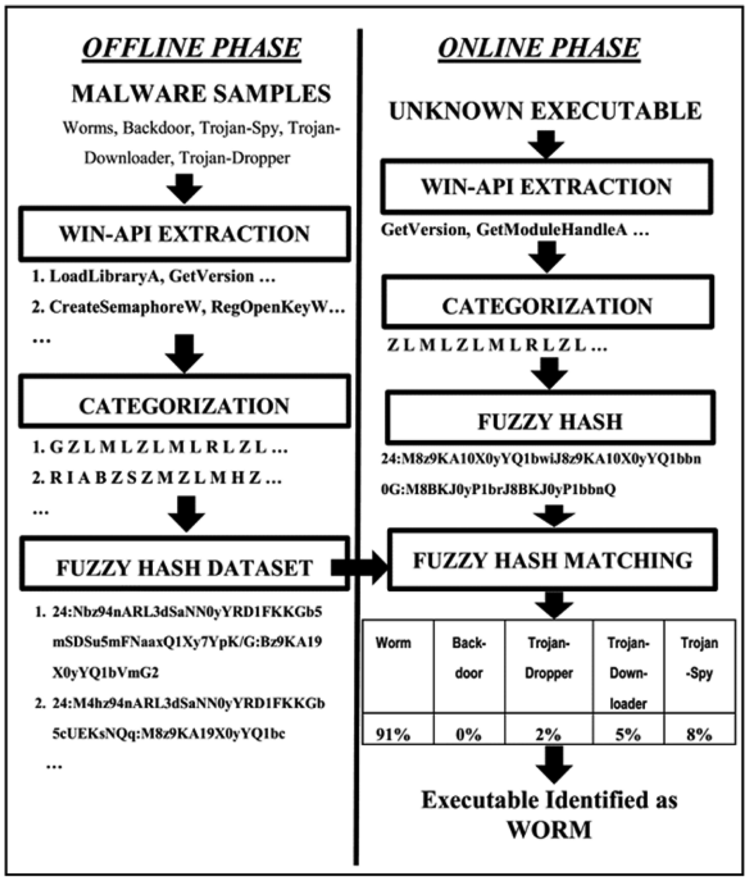
2.1 Overall Malware Classification and Characterization Framework

The Proposed Malware Characterization Framework is mainly using Win-API hooking technique for API call sequence extraction and Fuzzy Hashing technique for signature generation, matching and classification. To carry out this we have downloaded malware samples from available internet resources [17–19]. Further this malware dataset is tagged as per Kaspersky's Antivirus classification through free VirusTotal [20] scanning engine. In this work we have selected five classes of malware: Worm, Backdoor, Trojan-Downloader, Trojan-Dropper and Trojan-Spy. The reason for selecting these five classes is that we were able to get sufficient number of tagged samples for these categories.

Modules for API hooking and DLL injection were implemented in C language to extract the Win-API call sequences. In all a set of 534 Win-APIs were hooked. All the samples were run and their API call sequence was observed. Repeated consecutive API calls were removed while generating signatures to remove redundancies. To have higher level of abstraction, we bundled similar API-calls in one category. In all 26 such categories (A...Z) were created and all the API calls were replaced with the corresponding category. We generated these 26 categories based on the functionality of the Win-API calls.

We have selected 26 categories to categorize the Win-API set od 534 calls, as we observed that these are sufficient to capture the higher level of functionality description of any application. For example, category 'I' belongs to Registry write operations and include Win-APIs like RegSetValueA, RegSetValueW, RegSetValueExA, RegSetValueExW, RegCreateKeyA, RegCreateKeyW etc. Also we get support from many text mining tools for Alphabet domain (A...Z). We also observed that increasing the number of categories does not increase the accuracy of results.

Fuzzy Hashing algorithm ssdeep [21] has been applied to the categorized API call sequences to get the fuzzy hash signature of each malware sample. Thus, a Fuzzy hash signature repository has been created for all the samples of different classes. For a given test sample, we use the same procedure to extract its fuzzy hash signature. Further we apply fuzzy hash signature matching algorithm [21] between the given test sample and all the samples in the signature repository. These matched values were averaged for each of the five classes and the sample is classified to the highest matched class. We have also extracted unique Win-API call patterns for each category of Malware, which are in terms of sequences of these API calls. The schematic diagram of Malware Classification framework is shown in Figure 1. It shows two phases: Online Phase and Offline Phase. Offline Phase is the learning phase for our classifier. Here a database of Fuzzy hashes is prepared from the known malware samples. In the online phase a new malware is subjected to same procedure and its fuzzy



OFFLINE PHASE
MALWARE SAMPLES
Worms, Backdoor, Trojan-Spy, Trojan-Downloader, Trojan-Dropper
WIN-API EXTRACTION
1. LoadLibraryA, GetVersion …
2. CreateSemaphoreW, RegOpenKeyW…
…
CATEGORIZATION
1. G Z L M L Z L M L R L Z L …
2. R I A B Z S Z M Z L M H Z …
…
FUZZY HASH DATASET
1. 24:Nbz94nARL3dSaNN0yYRD1FKKGb5mSDSu5mFNaaxQ1Xy7YpK/G:Bz9KA19X0yYQ1bVmG2
2. 24:M4hz94nARL3dSaNN0yYRD1FKKGb5cUEKsNQq:M8z9KA19X0yYQ1bc
…
ONLINE PHASE
UNKNOWN EXECUTABLE
WIN-API EXTRACTION
GetVersion, GetModuleHandleA …
CATEGORIZATION
Z L M L Z L M L R L Z L …
FUZZY HASH
24:M8z9KA10X0yYQ1bwiJ8z9KA10X0yYQ1bbn0G:M8BKJ0yP1brJ8BKJ0yP1bbnQ
FUZZY HASH MATCHING
Executable Identified as WORM

Worm	Back-door	Trojan-Dropper	Trojan-Down-loader	Trojan-Spy
91%	0%	2%	5%	8%

Figure 1 Proposed malware classification framework.

hash is calculated. This fuzzy hash is then compared with the existing database of fuzzy hashes of known malware. Closest matching class of malware is then assigned to it. Implementation details regarding all the above-mentioned steps are given in the Sections 2.2 to 2.5.

2.2 Malware Dataset Preparation and Extraction of Win-API Calls

Malware samples were downloaded from internet [17–19]. These samples were not tagged, meaning no class specific information was available. But for training purpose we required tagged malware samples. To address this issue, we have used the online scanning services provided by Virustotal [20]. In particular, Kaspersky Anti-Virus engine classification was used to tag malware samples to appropriate classes.

Five Classes of Malware has been selected for further analysis: Worm, Backdoor, Trojan-Downloader, Trojan-Dropper and Trojan-Spy.

Trojan-Downloader, Trojan-Spy and Trojan-Dropper are the malware classes be-longing to Trojan family. These malware are disguised as legitimate software and perform malicious functionality in background. Trojan-Downloader downloads additional program (usually malware) from internet and infect the system. Trojan-Spy steals user's passwords and personal data by monitoring user's activity on computer. They generally perform keyboard hooking for stealing login or credit card information of user. Trojan-Dropper drops malicious programs to a system once executed. Unlike Trojan-Downloader, they contain these additional files compressed or encrypted inside their own body. Generally these additional files are stored in resource section of executable. Backdoors as name implies opens a back door (connection) for hacker or another program. Worm typically replicates itself and generally has intention of infecting whole internal network of user to perform malicious functionality.

For experimentation, we selected 520 samples for each class of malware: Worm, Backdoor, Trojan-Downloader, Trojan-Dropper and Trojan-Spy. We ran all these 2600 samples in virtual machine and extracted their Windows API (Win-API) call sequences. Here 2000 samples are taken for learning and 600 samples for testing purpose. The Win-API's provide access to the fundamental resources available to the Windows system. These are defined in Windows DLLs like kernel32.dll, advapi32.dll, gdi32.dll, comct32.dll, user32.dll, shell32.dll and ntdll.dll. There are multiple versions of Win-APIs which are represented with different suffixes: 'A', 'W', 'Ex', 'ExA' and 'ExW' based on Unicode and ANSI types. For our research we have selected 534 Win-API calls, which seems relevant for behaviour analysis of malware.

A C-program was written for User Level inline API hooking which uses Detours [22] library to extract the Win-API calls. Hooking is a technique used to modify the behaviour of API calls in operating system or applications or other software by intercepting their function calls or messages passed between User/Application layer and kernel layer. They are used for Debugging, Monitoring, and Intercepting messages and to extend functionality of any given binary.

Malware class namely Rootkits mainly employ hooking to hide itself in the system. Hooking can be done either in user-mode or in kernel of OS. Kernel hooking requires valid signed drivers and in-depth internal knowledge of OS kernel. User-Mode hooking is relatively simple and it is achieved by hooking Windows APIs or third party libraries. There are mainly two techniques to perform user land hooking namely Inline Hooking and Import Address Table (IAT) Hooking. Import address hooking patches the import address table of PE file to trick the application into execution of another function which carries out malicious functionality. Inline hooking is achieved by overwriting the beginning of DLL with a jump to the function which carries out malicious functionality. Inline hooking is considered more robust than IAT hooking as they do not have any problems related to DLL binding at run time. Also it can be used to hook other function calls, instead of only system calls. This technique is widely used by professionals.

The Win-API call sequences were extracted by running every sample for 30 seconds in the Virtual environment on Windows-XP. Consecutive same API calls were clubbed together to remove redundant information from API call sequences.

2.3 Categorization of Win-API Calls

We have categorized the total set of 534 Win-API calls into 26 Categories based upon the function these APIs are performing. These categories are developed by us and are shown in Table 1. This categorization has been done to club all the APIs used to achieve a higher level functionality into a single category. For example Win-API calls like Send, Recv, WSARecv and Connect are related to socket communication and hence are placed in Socket Communication category. These Categories are labelled by letters: A to Z. So each extracted Win-API call is replaced with its corresponding labels (A–Z). Thus every sequence of Win-API calls is mapped to a categorized sequence which is in terms of A to Z letters. The categorization is done with the aim of providing a higher level abstraction and also to make the model simple. Without this categorization we will have lot of redundancy in the extracted data, which will make feature extraction task more difficult.

2.4 Creating Fuzzy Hash Signatures

We have applied ssdeep [21] program to compute Context Triggered Piecewise Hash (CTPH), also called fuzzy hash, on the categorized API call sequences. The concept of fuzzy hashing has been used as it has the capability to compare two different samples and determine the level of similarity between them. Instead of generating a single hash for a file, piecewise hashing generates many hashes for a file based on different sections of the file. CTPH algorithm uses the rolling hash to determine the start and stop of each segment. CTPH Signature generation algorithm combines these section hashes in a particular way to generate the fuzzy hash of the file. Also, two inputs with sequence of identical bytes in between them can be identified using CTPH matching algorithm ssdeep [21]. We have selected this technique because CTPH can match inputs that have homologies and these sequences may be different in both content and length. As length of extracted Win-API sequences for each sample is different, fuzzy hashing technique suits us the most. These hashes constitute the signature repository. For our analysis, we have developed a repository of 2000 fuzzy hash signatures, 400 for each class.

Table 2 shows API call sequences and their fuzzy hash signatures for few samples of worm class. A signature of the file contains three parts (block size and two hashes) separated by colon letter. Block size ‘b’ of a file having size ‘n’ is calculated by using mathematical formula: $b = 3 \times 2^{\lfloor \log_2(\frac{n}{64}) \rfloor}$. First hash is computed with ‘b’ and other hash with ‘2b’. With two hashes in single signature one can compare two different signatures b$_x$ and b$_y$ if b$_x$=b$_y$ or b$_x$=2.b$_y$ or b$_y$=2*b$_x$.

2.5 Matching Fuzzy Hash Signatures

We have calculated and stored fuzzy hashes for samples of different classes of malware. Comparisons between different files can be performed via just fuzzy hashes, rather than actual files themselves. This is very helpful when looking at a new fuzzy hash to see if it might be related to any other fuzzy hashes in a database. For example, fuzzy hashes for both whitelisted and blacklisted can be generated and stored in a database.

Table 1 Categorization of Win-API Calls

Category	Code	No. of API	Some Examples
Input/output Create	A	14	CreatefileA, CreatePipe, CreateNamedPipeA
Input/output Open	B	10	OpenFile, OpenFileMappingA
Input/output Write	C	25	WriteFile, WriteConsoleW, WriteFileEx
Input/output Find	D	13	FindFirstFileA, FindNextFileW
Input/output Read	E	18	ReadFile, ReadFileEx, ReadConsoleA
Input/output Access	F	19	SetFileAttributesW, SetConsoleMode
Loading Library	G	7	LoadLibraryExW, FreeLibrary
Registry Read	H	15	RegOpenKeyExW, RegQueryValueA
Registry Write	I	13	RegSetValueA, RegSetValueW
COM/OLE/DDE	J	154	OleCreate, OleLoad, CoBuildVersion
Process Create	K	10	CreateProcessA, ShellExecute, WinExec
Process Read	L	33	GetCurrentThreadId, ReadProcessMemory
Process Write	M	10	WriteProcessMemory, VirtualAllocEx
Process Change	N	12	SetThreadContext, SetProcessAffinityMask
Process Exit	O	3	TerminateProcess, ExitProcess
Hooking	P	5	SetWindowsHookA, CallNextHookEx
Anti-Debugging	Q	4	IsDebuggerPresent, OutputDebugStringA
Synchronization	R	13	CreateMutexA, CreateSemaphoreW
Device Control	S	6	DeviceIoControl, GetDriveTypeW
Socket Comm.	T	70	Send, Recv, WSARecv, Connect
Network Information	U	17	Gethostbyname, InternetGetConnectedState
Internet Open/ Read	V	13	InternetOpenUrlA, InternetReadFile
Internet Write	W	2	InternetWriteFile, TransactNamedPipe
Win-Service Create	X	2	CreateServiceW, CreateServiceA
Win-Service Other	Y	11	StartServiceW, ChangeServiceConfigA
System Information	Z	35	GetSystemDirectoryW, GetSystemTime
Total APIs		**534**	

Ssdeep matching algorithm calculates the matching between fuzzy hashes of two different samples. This score is based on the edit distance algorithm. The string edit distance is a measure of how many edit operations are required to take one of the signatures and turn it into the other. Allowed operations during string matching are insertions (weight=1), deletions (weight=1), transposition (weight=5) and substitutions (weight=3). After matching a comparison score is generated between 0 and 100. We have used this matching score as malware classification criteria.

Table 2 Sample fuzzy hash signatures of worms

Malware	API Call Sequence	Fuzzy Hash
Worm 1	ZLMLZLMLRLZLZLSJLQBRLGSG SLZLDGZLRZJLSJSLHLHGQGLG ZBGZGZLZLZLMLMLMHZGML ZLZAFWMWMWMF...	24:Nbz94nARL3dSaNN0yYRD1F KKGb5mSDSu5mFNaaxQ1Xy 7YpK/G:Bz9KA19X0yYQ1bVmG
Worm 2	ZLMLZLMLRLZLZLSJLQLZBRL SLSLDGZLRZJLSJSLHLH GQGLGZBGZGZLZLZLMLM LMHZGMLZLZAFWMLMLMZF...	24:M4hz94nARL3d SaNN0yYRD1FKKGb5c UEKsNQq:M8z9KA19X0yYQ1bc
Worm 3	SLZLGLZLZLQBRLTLHLHZSHML ZIRLZCACSLSMGLSJSLHLHGQG LGZBGZGZLZLZLMLMLMHZGML ZLZAFLMLMLMZ...	24:lM2dV94nAsVPrr9WK0JPOEU f9uuSHS0uYC35AAW5AAtwYQ 4l3qNb2X:NP9KAMPr6JPOE8935 AAW5AAtwIlcc

3 Classification Results and Analysis

Our framework presently contains 2000 fuzzy hash signatures, 400 of each class. The ssdeep tool also has a fuzzy hash signature matching module which gives a matching score between 0 (totally different) and 100 (exactly same). For testing purpose, we took 120 samples for each category making a total of 600 samples. These samples were taken from different sources than the training samples. The test samples were run and their API call sequences were observed. Fuzzy hashes were calculated for all these categorized API call sequences. Fuzzy hash matching score was calculated between the test sample and all the samples of the training set. So average matching score is calculated for each class of test data. Further, we calculate matching score of test sample with each of the training sample to calculate the average matching score. Table 3 gives the consolidated average matching score between test samples and training samples for all the classes. It was observed that maximum matching is obtained between test samples and training samples of the same class. Diagonal entries in the table show the average matching between test and training samples of same class. These results gave us the confidence that fuzzy hashing can be used to classify the samples in different classes. So each of the 120 test samples were individually classified to the class based on maximum average matching score.

Figure 2 gives the details of the classification results for 120 test samples of each of the five categories.

Since there is no classification system available in literature for the above mentioned categories, so we have given comparison of our system with malware vs benign classifiers [11, 12], which are much simpler.

Table 3 Average matching score (0–100) of fuzzy hashes between different classes of test samples and training samples

	Dataset of 2000 Signatures (400*5)				
Test Samples (# 120)	Worm	Backdoor	Trojan-Dropper	Trojan-Downloader	Trojan-Spy
Worm	**25.28**	5.42	3.16	7.6	1.74
Backdoor	5.42	**22.14**	1.31	5.3	3.5
Trojan-Dropper	3.16	1.31	**24.77**	5.45	10.55
Trojan-Downloader	7.6	5.3	5.45	**27.73**	7.01
Trojan-Spy	1.74	3.5	10.55	7.01	**26.63**

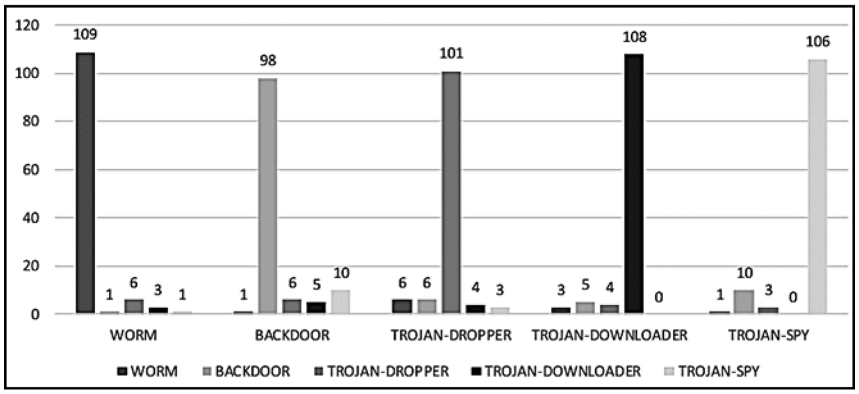


Figure 2 Classification results of 600 (120 * 5) test samples.

For this we have divided our 5-class problem into five 2-class problems, namely: Worm vs rest, Backdoor vs rest, Trojan-Dropper vs rest, Trojan-Downloader vs rest and Trojan-Spy vs rest. Table 4 gives the classification accuracy and FPR for these five 2-class problems, and Table 5 gives the accuracy & FPR for 2 class classifier problems (Malware vs Benign) [11, 12]. These classification results indicate that there exist class specific signatures for every class which can be extracted manually by thorough inspection. Thus some malware class specific signatures in terms of patterns were extracted. Table 6 gives few of the distinctive patterns extracted for each category. The table also shows the presence of these patterns in the other classes. These patterns are extracted using basic n-gram analysis which is based on exact matching algorithm. However many more patterns can be considered if we use approximate matching algorithms. It is the presence of these Win-API patterns, which aids in the fuzzy hash based classification of the five classes of Malware. Also our unique categorization of Win-API calls made this task easier and effective as now we have an abstract and simplified data to work on.

Table 4 Performance of our framework

Classification Problem	Classification Accuracy (%)	FPR
Worm vs rest	96.33	0.022
Backdoor vs rest	92.67	0.045
Trojan-Dropper vs rest	93.66	0.039
Trojan-Downloader vs rest	96	0.025
Trojan-Spy vs rest	95.33	0.029

Table 5 Performance of malware detection models given in [11, 12]

Malware Detection Model based on feature (Malware vs Benign)	Classification Accuracy (%)	FPR
Opcode n-gram + Byte code n-gram [1]	95	0.06
Opcode n-gram [2]	99	0.03
Opcode n-gram [3]	92	0.03
Byte Code n-gram + Opcode n-gram [4]	96	0.01
Opcode n-gram + API [6]	96.22	0.07
API + String + function length frequency [5]	97.05	0.055
Portable Executable Header + Strings [7]	93.7	0.15
System Call [23]	96.8	0.04
API Call with DNA sequence alignment [12]	99.8	-

Table 6 Presence (%) of some distinctive API patterns in malware

PATTERN	Worm	Backdoor	Trojan-Downloader	Trojan-Dropper	Trojan-Spy
EBMZRFZ RMHMHZH	0	0	**24.16**	0	0
LFAFECEAE	**23**	0	0	0	0
MLMLMLM LMLMLHDG	0	**58.95**	0	0	0
GLGLGLS MHMHM HM	0	**12.3**	0	0	0
ZDGLMH	3.75	0	**25.83**	0	2.5
LSLZXMXL	0	0	0	0	**33.33**
FLHZSRGL MLPLPLPL	0	2.1	0	0	**32.51**
LPLP LMLZJL	**26.67**	0	0	0	0
LPLJLJ	0	21.90	1.67	**46.15**	
LJZJHLJH	**12.3**	0	0	0	0
PZLSLZMLDG DGLDGZ	**16.7**	0	0	0	0
FZRFM JRIHLFIM	0.83	0	0	**21.97**	1.25

Careful examination of these patterns reveals that their presence is more frequent in malware belonging to same class. These patterns can be treated as run time signatures for each malware class and hence can be used as features by the anti-malware products.

4 Conclusion

Classification based on fuzzy-hash based matching score on Win-API call sequences gives good results to classify different kinds of malware. Five different classes of malware were analyzed: Worm, Backdoor, Trojan-Downloader, Trojan-Dropper and Trojan-Spy. The Win-APIs were categorized into 26 categories based upon their functionality and further analysis was carried out on these categorized sequences. With n-gram analysis on the categorized sequences, we were able to extract class specific patterns for all the five classes of Malware. Fuzzy hashes of these categorized sequences were calculated with ssdeep algorithm. Fuzzy hash based matching score was calculated between different categorized sequences. High fuzzy hash matching score was observed in samples belonging to same class. It was established that the fuzzy hash based matching score can used as classification criteria as it successfully captures the homologies in the behavior of the malware samples belonging to the same class.

5 Future Work

The proposed malware classification system will be extended to other malware classes. Fuzzy hash based matching scheme can be replaced with more sophisticated text pattern matching techniques. Extracted unique subsequences can also be considered as features for classification. Number of samples in each category will be increased for more accuracy. We propose to integrate all the activities into a single automated system which will check all running programs for malicious behaviour. At present API hooking has been done at User level which will be extended to Kernel level, if possible. A similar approach will be used to capture behaviour of applications based on other Operating systems like Linux, Android etc.

References

[1] Shafiq, M. Z., Tabish, S. M., Mirza, F., and Farooq, M. (2009). Pe-Miner: Mining structural information to detect malicious executable in real time. In *12th international symposium on recent advances in intrusion detection*.

[2] Moskovitch, R. et al., (2008). Unknown Malcode Detection Using OPCODE Representation. *Intelligence and Security Informatics*, Volume LNCS 5376, 204–215.

[3] Moskovitch, R. et al., (2008). Unknown Malcode Detection via text categorization and the imbalance problem. In *IEEE International Conference on Intelligence and Security Informatics*, 156–161.

[4] Santos, I. et al., (2013). Opcode sequences as representation of executables for data-mining based unknown malware detection. *Information Science*, 231, 64–82.

[5] Egele, M., Scholte, T., Kirda, E., Kruegel, C. (2012). A Survey on Automated Dynamic Malware Analysis Techniques and Tools. *ACM Computing Surveys*, 44(2) 1–42.

[6] Santos, I. et al., (2013) OPEM: A Static-Dynamic Approach for Machine-learning-based Malware Detection. In *International Conference CISIS12-ICEUTE12*, 189, 271–280.

[7] Ye, Y. et al., SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging. *Journal in Computer Virology*, 5(4), 283–293.

[8] Zolkipli, M. F., and Jantan, A. (2011) Approach for Malware Behavior Identification and Classification. In *3rd International Conference on Computer Research and Development*, Shanghai, 191–194.

[9] Islam, M. R., Tian, R., Batten, L., and Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Applications*, 36, 646–656.

[10] Gandotra, E., Bansal, D., and Sofat, S. (2014). Malware Analysis and Classification: A Survey. Journal of Information Security, 5, 56–64.

[11] Ranveer, S., and Hiray, S. (2015). Comparative Analysis of Feature Extraction Methods of Malware Detection. *International Journal of Computer Applications*, 120(5), 1–7.

[12] Youngjoon, K., Eunjin, K., and HuyKang, K. (2015). A Novel approach to detect Malware based on API call sequence analysis. *International Journal of Distributed Sensor Networks*, 4.

[13] Park, Y., Reeves, D., Mulukutla, V., and Sundaravel, B. (2010). Fast malware classification by automated behavioural graph matching, In *Sixth Annual Workshop on Cyber Security and Information Intelligence Research*. (P. 45). ACM.

[14] Nari, S., and Ghorbani, A. A. (2013). Automated Malware Classification based on Network Behavior, In *International Conference on Computing, Networking and Communications (ICNC)*, 642–647.

[15] Reina, A. et al., (2013). A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors. *ACM Eur. Work. Syst. Secur. (EuroSec)*, 1–6.
[16] Yu, W. et al., (2013). On behavior-based detection of malware on Android platform, *GLOBECOM - IEEE Glob. Telecommun. Conf.*, 814–819.
[17] VxVault, http://www.vxvault.net
[18] Vxheaven, http://www.vxheaven.org
[19] VirusSign, http://www.virussign.com
[20] VirusTotal, https://www.virustotal.com
[21] Kornblum, J., (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3, 91–97.
[22] Hunt, G., and Brubacher, D., (1999). Detours: Binary Interception of Win32 Functions. *3rd Conference on USENIX Windows NT Symposium*, 135–143.
[23] Firdausi, I. et al., (2010). Analysis of Machine Learning Techniques used in Behavior-Based Malware Detection. In *Second International Conference on Advances in Computing, Control and Telecommunication Technologies (ACT)*. IEEE. 201–203.

Biographies

Sanchit Gupta is graduated in Computer Science & Engineering from National Institute of Technology, Hamirpur. He joined Scientific Analysis Group (SAG) in 2005 and is presently working as Scientist ‘E’. He is working in the area of Malware Analysis and Software Security.

Harshit Sharma has completed post-graduation in MTech. Computer Science & Engineering (Spl. In Cyber Security) from Sharda University, Greater Noida in May 2018. His area of interests is Digital Forensics and Malware Analysis.

Sarvjeet Kaur has done M.Sc (Computer Science) from DAVV, Indore. She also did M.S. (Software System) from BITS, Pilani in 2010. She joined Scientific Analysis Group (SAG) in 1991 and is presently working as Scientist ‘F’ and heading the Software Security testing Group. Her area of interests are Software Security and Malware Analysis.