Exploring Timeline-Based Malware Classification

Rafiqul Islam, Irfan Altas, Md. Saiful Islam

Exploring Timeline-Based Malware Classification

[1]Rafiqul Islam, [1]Irfan Altas, and [1,2]Md. Saiful Islam

[1]Charles Sturt University, Australia
{mislam, ialtas}@csu.edu.au

[2]Swinburne University of Technology, Australia
mdsaifulislam@swin.edu.au

Abstract. Over the decades or so, Anti-Malware (AM) communities have been faced with a substantial increase in malware activity, including the development of ever-more-sophisticated methods of evading detection. Researchers have argued that an AM strategy which is successful in a given time period cannot work at a much later date due to the changes in malware design. Despite this argument, in this paper, we convincingly demonstrate a malware detection approach, which retains high accuracy over an extended time period. To the best of our knowledge, this work is the first to examine malware executables collected over a span of 10 years. By combining both static and dynamic features of malware and cleanware, and accumulating these features over intervals in the 10-year period in our test, we construct a high accuracy malware detection method which retains almost steady accuracy over the period. While the trend is a slight down, our results strongly support the hypothesis that perhaps it is possible to develop a malware detection strategy that can work well enough into the future.

Keywords: Timeline, Malware Detection, Static and Dynamic Features.

1 Introduction

Malware is one of the biggest challenges all over the world nowadays among the Internet users. Malware writers use various obfuscation techniques to transform a malicious program into undetectable variants with the same core functionalities of the parent malware program [1], [4], [7], [13], [15], [21]. Anti-Malware (AM) communities are trying hard to combat malware obfuscation techniques adopted by the malware writers by discovering the behavioral patterns of their parent malwares. However, one of the biggest challenges is that an AM strategy that has been found to be successful in a given time period cannot work at a much later time. This philosophy is supported by the works found in [1], [2], [9], [15], [16], [20] and [21], which indicates that current techniques fail to find the distinctive patterns of malicious software which can be used to identify future malwares. The argument is that malware evolves with time and eventually becomes unrecognizable from the original form; in addition completely new malware is designed which is unlike any known malware and so would not be detected by anti-virus software constructed to detect known types of malware. In fact, the assumption that malwares which are completely unlike earlier malwares designed on a major scale is known to be false as indicated by the statistics in [3] showing that barely 25% of malwares found in 2006 are not variants of known malwares.

Despite the strong support in the literature of the assumption that malware detection methods cannot easily detect future malware, in this paper, we convincingly demonstrate that perhaps, it is possible to develop a malware detection strategy which can retain high accuracy over an extended time period. To the best of our knowledge, this paper is the first to examine malware executables collected over a span of 10 years. The key contributions of this paper are two-folds:

(a) A novel approach to feature collection by accumulating malware features over time segments of the 10 years span.
(b) A novel malware detection method retaining steady accuracy over an extended period of time.

The rest of the paper is organized as follows: Section 2 provides a review of the literature and Section 3 describes the set-up for the testing. In Section 4, we provide a detailed discussion of the experiment and present the results and in Section 5, we discuss the analysis and its implications for future work.

2 Related Work

A substantial research has been done on malware classification and detection. In this section, we present only a few of the existing works that are either closely related to or motivate us to conduct our research in this paper. The study in [20] investigates malicious attacks on several websites by creating web honey pots and collecting website-based malware executables over a period of five months. In their study, they collect and analyze malware samples using 6 different antivirus programs, and conduct the same experiment four months later using the updated versions of the 6 programs to determine their efficacy. Additionally, the work of Rajab et al. [10] also focuses on mitigating web-based malware. They study a dataset collected over a period of four years and demonstrate that existing malware characteristic can aid in detecting future malware. Both aforementioned works demonstrate that, some anti-virus software can significantly improve detection rates with training on older malware.

The research conducted by Rosyid et al. [11] is focused on detecting malicious attack patterns in botnets attacking a honeypot during the year 2009. After extracting the log files of malware sequences, they then apply the *PrefixSpan* algorithm to discover subsequence patterns. The authors extend their work by identifying attack patterns based on IP address and timestamp. The authors argue that the signature of a single malware file is not enough to detect the complex variants of the attacks by botnets. In [1], the authors apply a dynamic method for classifying malware, considering the interactions between the operating system and malicious programs as behavioral features. In their study they use three different evaluation techniques: completeness, conciseness, and consistency. The authors mention that one limitation of their methodology is the failure to "detect fine-grained characteristics of the observed behaviors".

Another group of researchers, [6], build their malware detection and classification framework based on comparisons of extracted strings using static analysis. They claim that the similarity between two files can be determined by comparing the character strings, which in turn is used to identify and determine whether the two instances are variants. The authors present a three-step methodology of extraction, refinement and comparison. The authors show that if a mutated instance of malware is detected, it is reflected as a huge peak under the respective malware family.

In [5], the authors use a combination of static and dynamic analysis to achieve a high level of accuracy over an 8 year time period. We are not the first to integrate dynamic with static features however (see for example [12]), though we are the first to use this method applied over a long time period. In order to understand the evolution of malware over a long period of time and its effects on future malware, in this paper, we consider two types of analysis: static and dynamic, as these features are predominant for malware analysis in the literature.

3 Experimental Setup

3.1 The methodology

Static and dynamic analyses are two of the most popular forms of malware analysis techniques predominant in the literature [1], [5], [6], [9], [12], [17], [18], [19], [20]. However, each of these analysis techniques comes with its own merits and demerits. Static analysis can analyze a wide spectrum of possible execution paths of an executable, thus providing a good global view of the whole executable and of the entire program logic without running it. But, static analysis is susceptible to inaccuracies due to obfuscation and polymorphic techniques.

On the otherhand, dynamic analysis monitors the behavior of the binary executable file during its execution, which enables it to collect a profile of the operations performed by the binary thus offering potentially greater insight into the code itself. The main limitation of dynamic analysis is that analysis results are only based on malware behavior during a specific execution run. Since some of the malware's behaviour may be triggered only under specific conditions, such behaviour would be easy to miss with only a single execution.

In our experiments, we extract both static and dynamic features from the malware and cleanware files collected over the 10 years period and learn our classifier to detect future malwares. More specifically, we extract from each executable (a) static features: printable string information (PSI) and function length frequency (FLF), (b) dynamic features: API calls including their parameters and (c) integrated features: a combination of the two static and the dynamic features. The WEKA library of data mining algorithms [8] is used to learn the classifiers and derive the detection results based on the extracted feature vectors as input.

3.2 Data (malware and cleanware) collection

The malware executables used in the experiment were collected from CA's VET Zoo[1] over a span of 8 years (2002-2010) and we collect (2011-2012) manually from open sources (www.offensivecomputing.net, http://www.virussign.com); the clean-ware executables were collected manually from various versions of Win32 based systems. Fig. 1 indicates the dates at which malware files were collected.

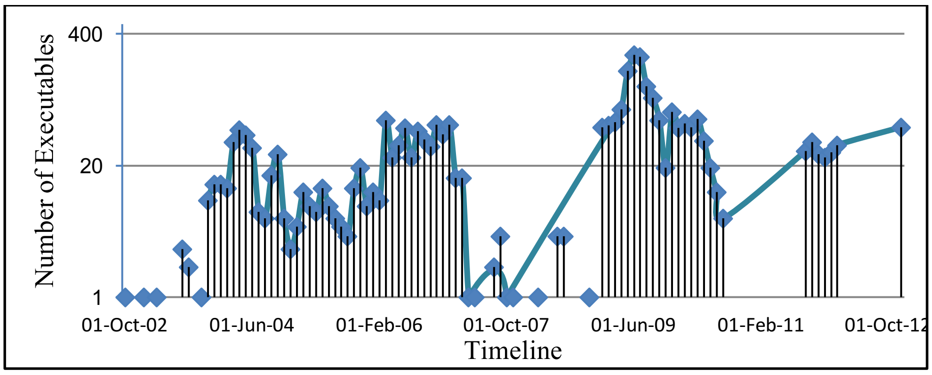


Fig. 1. Collection of malware executables from 2002 to 2012

The total numbers of malware and cleanware executables, used in our experiments, are 2617 and 541 respectively. Table 1 shows the executables family by family.

3.3 Timeline data preparation

The date of a malware file was associated with the file when the file was collected. We exported all files, along with their dates, into our Ida2DBMS schema [18] and based on the dates broke the data into groups as described in Fig. 2.

To generate groups of malware for use in the testing, we begin with the earliest malware and add month by month across the timeline until all data are grouped. As the first data group, MG_1, we take the earliest-dated 10% of the files. There are 262 executables in this group which covers the period from October 2002 to December 2004. The second data group, MG_2, comprises the data collected during the period October 2002 to January 2005, and so on. When too few files appear in a subsequent month to justify including that month as a group, we jump to the following month. In all, this results in 65 malware data groups which are labeled as MG_1, MG_2,…,MG_{65}. Fig. 2 indicates the spread of malware across the sixty five groups with each bar corresponding to a group.

Throughout the test, the set of 541 WIN32 cleanware files is treated as a single group, cleanware group (*CG)*. However, when it is tested against a particular malware group, depending on the comparative size of the two groups, the cleanware group may be divided into subgroups.

[1] www.ca.com.au

Table 1. Experimental set of 3158 files

Type		Family	Detection Date Starting ⇒ ending	Number of Executables	Number of Instances
Malware	Trojan	Bambo	2003-07 ⇒ 2006-01	44	5100
		Boxed	2004-06 ⇒ 2007-09	178	56662
		Alureon	2005-05 ⇒ 2007-11	41	7635
		Robknot	2005-10 ⇒ 2007-08	78	10411
		Clagger	2005-11 ⇒ 2007-01	44	4520
		Robzips	2006-03 ⇒ 2007-08	72	6549
		Tracur	2011-08⇒2011-11	42	7365
		Cridex	2011-10 ⇒2012-02	56	8692
	Worms	SillyDl	2009-01 ⇒ 2010-08	439	56933
		Vundo	2009-01 ⇒ 2010-08	80	1660
		Lefgroo	2012-12⇒2012-12	121	35421
		Frethog	2009-01 ⇒ 2010-08	174	28042
		SillyAutorun	2009-01 ⇒ 2010-05	87	9965
	Virus	Gamepass	2009-01 ⇒ 2010-07	179	23730
		Bancos	2009-01 ⇒ 2010-07	446	89554
		Adclicker	2009-01 ⇒ 2010-08	65	11637
		Banker	2009-01⇒ 2010-06	47	12112
		Agobot	2002-10 ⇒ 2006-04	283	216430
		Looked	2003-07 ⇒ 2006-09	66	36644
		Emerleox	2006-11 ⇒ 2008-11	75	61242
Total of malware files			***2002* ⇒ 2012**	**2617**	**690304**
Cleanware				**541**	81154
TOTAL				**3158**	**771458**

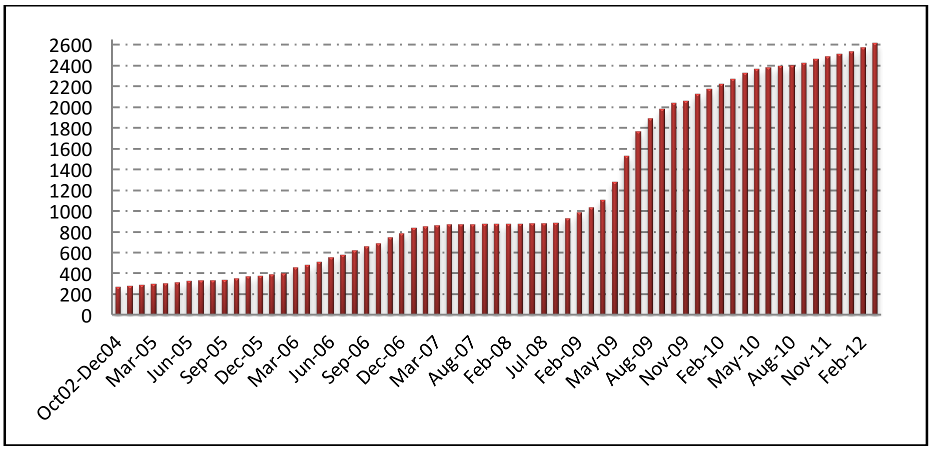


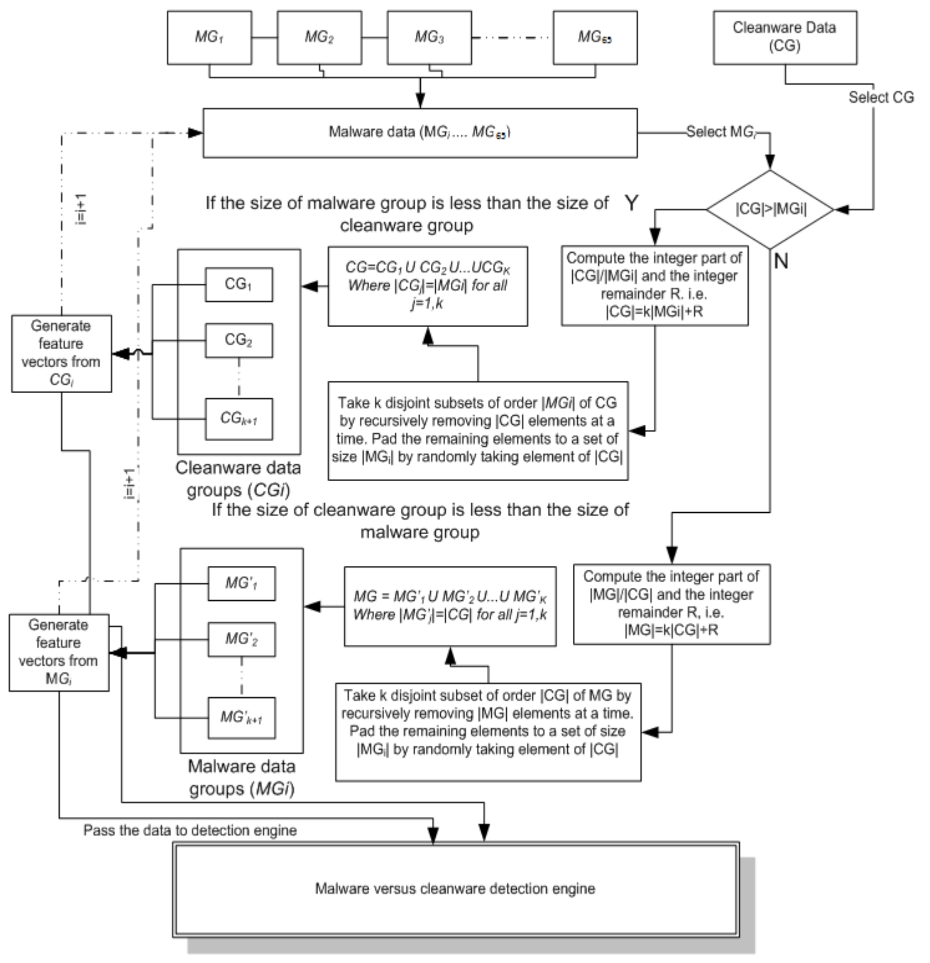
Fig. 2. Number of malware executables accumulated by date

3.4 Cumulative feature vector (CFV) generation

To test malware against cleanware, we fix a malware group and use an equal portion of malware and of cleanware data. Fig. 3 shows the data preparation process. The selected malware group MG_i is compared with CG. If $|MG_i|$ is smaller than $|CG|$ then we compute the integer part of $|CG|/|MG_i|$ and the integer reminder $0 \leq R < |MG_i|$ as in $|CG| = k\,|MG_i| + R$, for some positive integer k.

We then divide CG into k disjoint groups of equal size. If $R > 0$, then the remaining elements must be padded out to a (k+1)'st group CG_{k+1}. However, if R = 0, this set is empty and is not used.

If $|MG_i|$ is bigger than $|CG|$ then we compute the integer part of $|MG_i|/|CG|$ and proceed in the same way. This procedure is repeated for every malware group.



MG1
MG2
MG3
MG65
Cleanware Data (CG)
Select CG
Malware data (MGi.... MG65)
Select MGi
|CG|>|MGi|
Y
N
i=i+1
If the size of malware group is less than the size of cleanware group
Compute the integer part of |CG|/|MGi| and the integer remainder R. i.e. |CG|=k|MGi|+R
CG=CG1 U CG2 U...UCGK Where |CGj|=|MGi| for all j=1,k
Take k disjoint subsets of order |MGi| of CG by recursively removing |CG| elements at a time. Pad the remaining elements to a set of size |MGi| by randomly taking element of |CG|
CG1
CG2
CGk+1
Cleanware data groups (CGi)
Generate feature vectors from CGi
If the size of cleanware group is less than the size of malware group
Compute the integer part of |MG|/|CG| and the integer remainder R, i.e. |MG|=k|CG|+R
MG = MG'1 U MG'2 U...U MG'K Where |MG'j|=|CG| for all j=1,k
Take k disjoint subset of order |CG| of MG by recursively removing |MG| elements at a time. Pad the remaining elements to a set of size |MGi| by randomly taking element of |CG|
MG'1
MG'2
MG'k+1
Malware data groups (MGi)
Generate feature vectors from MGi
Pass the data to detection engine
Malware versus cleanware detection engine

Fig. 3. Cumulative Feature Vector (CFV) generation

The pseudocode of our proposed cumulative feature vector (CFV) generation and testing algorithm is given below:

```
Step 1:  First we import the data from our database and group it according to system date. In our first data-group we called "$MG_1$", which is 10% of total malware from early malware executables. Then, we do remaining groups as month by month.
Step 2:  We add the data in increasing order as mentioned earlier, so that each data-group $MG_i$ is the $MG_i = MG_i + MG_{i-1}$.
Step 3:  Since our test is based on malware versus cleanware test, we again divide each data-group MGi into subgroups, same size of cleanware, if the malware size is bigger; and vice versa for cleanware, if the cleanware size is bigger.
Step 4:  Select a data-group (both from malware and cleanware) and extract the feature information (explained in the following section).
Step 5:  Create CFV and construct the .arff file (WEKA) format.
Step 6:  Select the data group and split it into n (n = 10) folds.
Step 7:  Build training data using n-1 folds (90% of the total data) and the remaining fold as test data (10% of the total data). The detail can be found in Section 3.6.
Step 8:  Call WEKA libraries to train the classifier using training data.
Step 9:  Evaluate the test data set.
Step 10: Repeat until finish for all n folds (each fold should be used as test data and the remaining folds as training data).
Step 11: Repeat for other classifiers.
Step 12: Repeat until finish all data groups.
```

3.5 Feature Vector generation

In our experiment, we use both static and dynamic features as explained before. We also use a combined version of these two features for the purpose of detecting future malwares.

Static features. From each of the executables, we extract two static features, function length frequency (FLF) and printable string information (PSI). These features are extracted from unpacked malware executables by means of a command line AV engine. The AV engine identifies and unpacks the packed executables in batch mode and identifies functions and printable strings.

To extract FLF features, we follow the methodology described in [17]. However, in order to determine an appropriate number of intervals, we follow Sturges' well-established statistical formula [14], which recommends the use of approximately $1+\log_2(n)$ bins, where n represents the number of instances in the experiment. Based on our value of n = 771458 we use 20 bins.

As an example, consider an executable file with 22 functions which have the following lengths in bytes, presented in increasing order of size: 5, 6, 12, 12, 15, 18, 18, 50, 50, 130, 210, 360, 410, 448, 546, 544, 728, 848, 1344, 1538, 3138, 4632. For the purposes of illustration, we create 10 bins of function length ranges. The distribution of lengths across the bins is as shown in Table 2.

This produces a vector of length 10 using the entries in the second column of Table 2: (0.0, 2.0, 5.0, 2.0, 1.0, 4.0, 4.0, 3.0, 1.0, 0.0) which corresponds to the function length frequency for the file chosen.

In extracting PSI features, we use the methodology of [18] for generating vectors to use in the classification process. We illustrate this with an example. Consider a global ordered string list containing 10 distinct strings:

{"LoadMemory","GetNextFileA", "FindLastFileA" "GetProcAddress", "RegQueryValueExW", "CreateFileW", "OpenFile", "FindFirstFileA", "FindNextFileA", "CopyMemory"}

Suppose that a particular executable has the following set of printable strings:

{"GetProcAddress", "RegQueryValueExW","CreateFileW","GetProcAddress"}

The PSI vector for this executable file records, first of all, the total number of distinct strings in the file followed by a binary report on the presence of each string in the global list, where a 'true' represents the fact that the string is present and a 'false' that it is not. Table 3 presents the corresponding data for this executable file. The vector for this example becomes (4, false, false, false, true, true, true, false, false, false, false).

Table 2. FLF bin distribution example

FLF Bin distribution	
Length of functions	**FLF Vectors**
1-2	0.0
3-8	2.0
9-21	5.0
22-59	2.0
60-166	1.0
167-464	4.0
465-1291	4.0
1292-3593	3.0
3594-9999	1.0
>=10000	0.0

Table 3. Example of PSI vector generation

PSI Vector generation	
Printable string	**PSI Vector**
String number	4
LoadMemory	false
GetNextFileA	false
FindLastFileA	false
GetProcAddress	true
RegQueryValueExW	true
CreateFileW	true
OpenFile	false
FindFirstFileA	false
FindNextFielA	false
CopyMemory	false

Dynamic Features. To extract this feature, we follow the methodology to generate dynamic logs, described in [19]. For generating a normalized feature vector from dynamic log files we construct a global feature list from all extracted API calls and API parameters. We treat the functions and parameters as separate entities as they may separately affect the ability to detect and identify the executable. For illustration, consider the following global API feature list:

{"RegOpenKeyEx","RegQueryValueExW","Compositing","RegOpenKeyExW","0x54", "ControlPanel\Desktop","LameButtonText","LoadLibraryW",".\UxTheme.dll","LoadLibrary ExW", "MessageBoxW"}.

We now list the distinct features in the global list and generate a vector for the executable based on the frequency of these features. Table 4 shows the distinct global feature list with corresponding frequencies for this particular example and the corresponding feature vector is (0, 2, 1, 1, 1, 1, 1, 1, 2, 1, 0).

Table 4. Example of dynamic feature vector generation

Dynamic feature vector generation	
Global feature	Frequency
RegOpenKeyEx	0
RegQueryValueExW	2
Compositing	1
RegOpenKeyExW	1
0x54	1
ControlPanel\Desktop	1
LameButtonText	1
LoadLibraryW	1
.\UxTheme.dll	2
LoadLibraryExW	2
MessageBoxW	0

3.6 Detection Method

In our classification process, we input the generated feature vectors into the WEKA classification system [8] for which we have developed an interface. In all of our experiments, 10-fold cross validation is applied to ensure a thorough mixing of the sample data and thereby, reducing the biasness as much as possible. In this procedure, we first select one group of malware data from a particular data set and divide it into ten portions of equal size. Then we select cleanware data of the same size as the group of malware data and also divide it into ten portions. The portions are, then, tested against each other. The whole process is repeated for each group and we then calculate average classification results. In our test, we have multiple tests within each group. The following Fig. 4 shows the number of tests within a period range.

To establish the training set, our detection engine takes nine portions from each of the malware and cleanware. The remaining portions from both malware and cleanware are used for the testing set. As is customary, the training set is used to establish the model and the testing set is used to validate it. The whole process is repeated so that every portion of both malware and cleanware is chosen as testing data. The results are then averaged. In order to ensure that the input vectors are trained and tested over a broad spectrum of classifiers, we chose the following four classifiers from WEKA [8] as they represent differing approaches to statistical analysis of data: Support Vector Machines (SVM), Random Forest (RF), Decision Table (DT) and IB1.

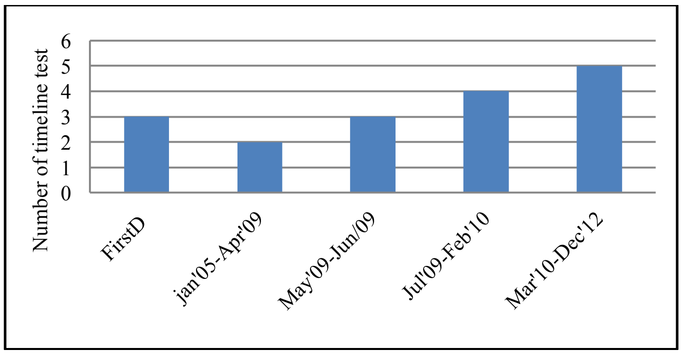


Fig. 4. Number of test within the group of data set

4 Experiments and Results

We have run the entire experiment using each of the four base classifiers SVM, IB1, DT and RF mentioned in Section 3.6. In addition, each test was run five times and the results averaged in order to ensure that any anomalies in the experimental set-up were discounted. The following sections present our empirical results.

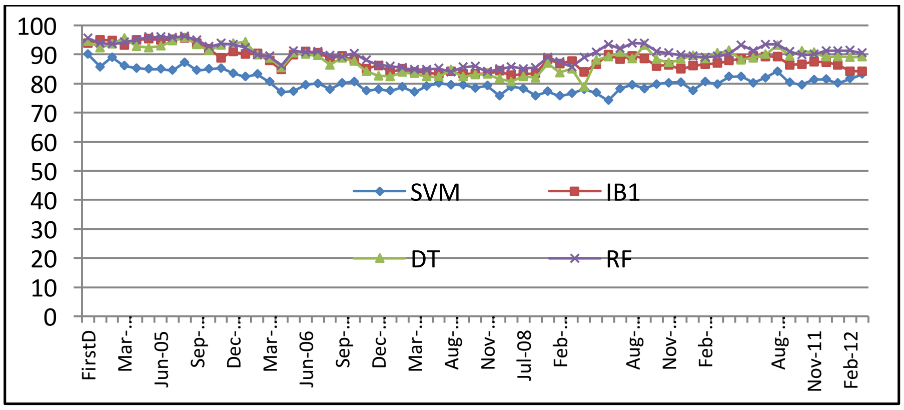


Fig. 5. Timeline results based on FLF features

4.1 Timeline classification results using FLF features

As mentioned in Section 3.5, the total number of FLF features used is fixed at 20 (bins) throughout the tests. Fig. 5 shows the result over the 10 year timeline. The *x*-axis shows the timeline data group and the *y*-axis shows the detection ratio of malware. It is clear that we achieve better detection accuracy for early malware compared to that for later malware. IB1 and RF give consistently better results across the timeline. SVM gives the worst performance, likely due to the very small feature set used in the experiment given that it is designed to handle large feature spaces. All classifiers except SVM maintain their accuracies above 80% throughout the timeline. Specifically, RF maintains better accuracy throughout the timeline compared to other classifiers.

4.2 Timeline classification results using PSI features

The number of PSI features used for each data group grows across the timeline, varying from approximately 800 to 2085. Fig. 6 shows the detection results using the PSI feature set. All classifiers except DT maintains their accuracies above 80% using PSI features. However, once again RF maintains better accuracy compared to the other classifiers throughout the timeline.

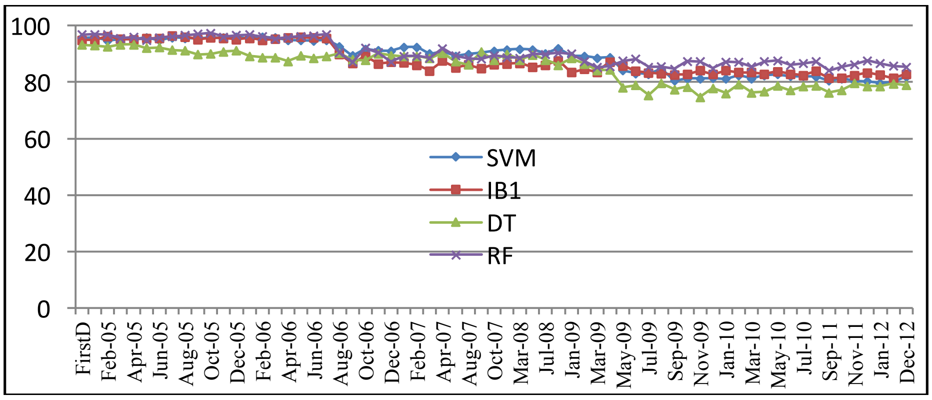


Fig. 6. Timeline results based on PSI features

4.3 Timeline classification results using dynamic features

The number of features used in this test increases from approximately 2600 to 8800 over the timeline. Fig. 7 shows the detection accuracy of the dynamic feature set which is based on API calls and API parameters.

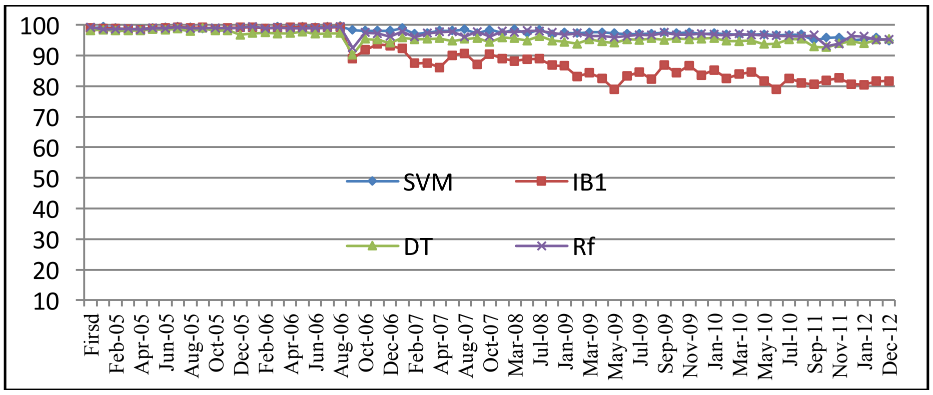


Fig. 7. Timeline results based on dynamic test

For all but the IB1 classifier, the dynamic feature results give almost consistent and better performance compared to either of the static features (i.e., FLF and PSI). With a large feature set, IB1 is inclined to make errors when building the training sets, which is likely to account for the poor performance here.

In Table 5, we have summarized the detection results for the last group, the 65th group of our data set, which includes all of the executables, showing the false positive (FP) and false negative (FN) rates and also the accuracies. Note the very poor accuracy of SVM with PSI while, with IB1, FLF has slightly better accuracy than the dynamic and PSI test, and with DT and RF, the results for dynamics features are better. It is thus difficult to argue that FLF should be deleted from the test altogether. Additionally, the false negatives and false positives of all tests in the case of IB1and DT are much higher than other two classifiers. However, the dynamic features give better performances comparing all parameters.

Table 5. Comparison of the performances of the tested classifiers for the 65th group

Features	**SVM**			**IB1**			**DT**			**RF**		
	FP	**FN**	**Acc**	**FP**	**FN**	**Acc**	**FP**	**FN**	**Acc**	**FP**	**FN**	**Acc**
FLF	0.07	0.66	83.26	0.14	0.25	84.23	0.17	0.28	89.17	0.03	0.11	90.51
PSI	0.12	0.25	81.98	0.1	0.23	82.92	0.15	0.24	78.87	0.12	0.21	85.33
Dynamic	0.01	0.12	94.81	0.17	0.16	81.65	0.02	0.08	95.35	0.03	0.04	95.29

5 Conclusion and Future Work

In this paper, we have presented a cumulative timeline approach for identifying malware from cleanware and demonstrated that perhaps it is possible to develop a malware detection strategy that can retains high accuracy over the long time period. The results presented in Section 4 indicate that our method retains fairly consistent accuracy over the 10 year period.

Our approach to feature collection is novel in that we accumulate the features over time segments of the 10 year span. In progressively adding malware over the time period, we thereby strengthen the accuracy of the test. The implication for anti-virus engines is that they are then able to use previously detected malware to provide features based on which to test new executables.

The results presented in section 4 indicate that no one feature type is the most significant for the specific classifier over the 10 year span. In our experiment, the static (FLF and PSI) and dynamic features act independently. Therefore, it is expected that combining static and dynamic features in an integrated manner could give a better detection rate. We would like to explore in our future work.

References

1. Bailey, M., Oberheide, J., Andersen, J., Mao, Z., Jahanian, F., Nazario, J.: Automated classification and analysis of internet malware. In: RAID. pp. 178-197 (2007)
2. Barford, P., Yegneswaran, V.: An inside look at botnets. In: Christodorescu, M., Jha, S., Maughan, D., Song, D., Wang, C. (eds.) Malware Detection, Advances in Information Security 27: 171-191, Springer US (2007)
3. Braverman, M., Williams, J., Mador, Z.: Microsoft security intelligence report January-June 2006. Accessed on 19th May 2011 (http://www.microsoft.com/en-us/download/details.aspx?id=5454)
4. Fossi, M., Johnson, E., Mack, T., Turner, D., Blackbird, J., Low, M. K., Adams, T., Mckinney, D., Entwisle, S., Laucht, M. P., Wueest, C., Wood, P., Bleaken, D., Ahmad, G., Kemp, D., Samnani: Symantec global internet security threat report trends for 2009. Technical report (2009)
5. Islam, R., Tian, R., Batten, L.M., Versteeg, S.: Classification of malware based on integrated static and dynamic features. J. Network and Computer Applications 36(2): 646–656 (2013)
6. Lee, J., Im, C., Jeong, H.: A study of malware detection and classification by comparing extracted strings. In: Proc. of the 5th International Conference on Ubiquitous Information Management and Communication. pp. 75:1-75:4. ACM, New York, NY, USA (2011)
7. Marcus, D., Greve, P., Masiello, S., Scharoun, D.: Mcafee threats report: Third quarter 2009. Technical report, Mcafee (2009)
8. Mark, H. Eibe, F., Geoffrey, H., Bernhard, P., Peter, R., Ian, H. W.: The WEKA Data Mining Software: An Update. SIGKDD Explorations 11(1): 10-18 (2009)
9. Nair, V.P., Jain, H., Golecha, Y.K., Gaur, M.S., Laxmi, V.: Medusa: Metamorphic malware dynamic analysis using signature from API. In: Proceedings of the 3rd international conference on Security of information and networks. pp. 263-269. SIN '10, ACM, New York, NY, USA (2010)
10. Rajab, M., Ballard, L., Jagpal, N., Mavrommatis, P., Nojiri, D., Provos, N., Schmidt, L.: Trends in circumventing web-malware detection. Google Tech. Rep. (2011)
11. Rosyid, N.R., Ohrui, M., Kikuchi, H., Sooraksa, P., Terada, M.: A discovery of sequential attack patterns of malware in botnets. In: SMC. pp. 2564-2570 (2010)
12. Roundy, K.A., Miller, B.P.: Hybrid analysis and control of malware. In: Proc. of the 13th International Conference on Recent advances in intrusion detection. pp. 317-338. Springer-Verlag, Berlin, Heidelberg (2010)
13. Soltani, S., Khayam, S.A., Radha, H.: Detecting malware outbreaks using a statistical model of blackhole traffic. In: ICC. pp. 1593-1597 (2008)
14. Sturges, H.A.: The choice of a class interval. Journal of the American Statistical Association 21(153): 65–66, 1926.
15. Sukwong, O., Kim, H., Hoe, J.: Commercial antivirus software effectiveness: An empirical study. Computer 44(3), 63-70 (2011)
16. Tang, H., Zhu, B., Ren, K.: A new approach to malware detection. In: Park, J., Chen, H.H., Atiquzzaman, M., Lee, C., hoon Kim, T., Yeo, S.S. (eds.) Advances in Information Security and Assurance, Lecture Notes in Computer Science, vol. 5576, pp. 229-238. Springer Berlin Heidelberg (2009)
17. Tian, R., Batten, L.M., Versteeg, S.C.: Function length as a tool for malware classification. In: Proc. of the 3rd International Conference on Malicious and Unwanted Software: MALWARE 2008. pp. 69-76 (2008)

18. Tian, R., Batten, L., Islam, R., Versteeg, S.: An automated classification system based on the strings of Trojan and virus families. In: Proc. of the 4th Inter-national Conference on Malicious and Unwanted Software: MALWARE 2009. pp. 23-30 (2009)
19. Tian, R., Islam, R., Batten, L., Versteeg, S.: Differentiating malware from cleanware using behavioural analysis. In: Proceedings of the 5th International Conference on Malicious and Unwanted Software : MALWARE 2010 (2010)
20. Yagi, T., Tanimoto, N., Hariu, T., Itoh, M.: Investigation and analysis of malware on web-sites. In: Lucca, G.A.D., Kienle, H.M. (eds.) WSE. pp. 73-81. IEEE Computer Society (2010)
21. You, I., Yim, K.: Malware obfuscation techniques: A brief survey. In: BWCCA. pp. 297-300 (2010)