The ISC Int'l Journal of Information Security

ISeCure

January 2013, Volume 5, Number 1 (pp. 71–82)

http://www.isecure-journal.org

DyVSoR: Dynamic Malware Detection Based on Extracting Patterns from Value Sets of Registers☆

Mahboobe Ghiasi [1], Ashkan Sami [1,*], and Zahra Salehi [1]

[1] *Computer Science and Engineering and Information Technology Department, Shiraz University, Shiraz, Iran*

ARTICLE INFO.

Article history:
Received: 2 January 2013
Revised: 28 August 2013
Accepted: 3 September 2013
Published Online: 25 October 2013

Keywords:
Malware Detection, API Call, Dynamic Analysis, CPU Register Values, x86 Registers Values.

ABSTRACT

To control the exponential growth of malware files, security analysts pursue dynamic approaches that automatically identify and analyze malicious software samples. Obfuscation and polymorphism employed by malwares make it difficult for signature-based systems to detect sophisticated malware files. The dynamic analysis or run-time behavior provides a better technique to identify the threat. In this paper, a dynamic approach is proposed in order to extract features from binaries. The run-time behavior of the binary files were found and recorded using a homemade tool that provides a controlled environment. The approach based on DyVSoR assumes that the run-time behavior of each binary can be represented by the values of registers. A method to compute the similarity between two binaries based on the value sets of the registers is presented. Hence, the values are traced before and after invoked API calls in each binary and mapped to some vectors. To detect an unknown file, it is enough to compare it with dataset binaries by computing the distance between registers, content of this file and all binaries. This method could detect malicious samples with 96.1% accuracy and 4% false positive rate. The list of execution traces and the dataset are reachable at: http://home.shirazu.ac.ir/∼ sami/malware

1 Introduction

Malware detection is a system that detects a program which has malicious intent [1]. Due to the exponential rise in unique variations of malware [2–9] and the weakness of security tools, a malware author may intend to attack organizations to steal the crucial financial data of banking credentials [10]. In 2012, cybercriminals used 500,000 payment cards to perform fraudulent transactions worth over US$25 million [11].

In addition, malware writers or cyber offenders used malwares as armaments to carry on their attacks [8]. A 42% growth in targeted attacks is reported by Symantec in 2012 [12]. Besides, More complicated and dangerous malwares like Stuxnet in 2010, Duqu in 2011 and Flamer and Disttrack in 2012 were used to specifically target some industrial control systems such as Iranians nuclear equipments and Saudi oil firms [9, 12–14]. Mcafee also reported one of the largest cybercrime hauls in history in 2012 that infected 11 million computers and caused the loss of over US$850 million [11]. On the other hand, the impact of cybercrime is worrying with 556 million victims per year, a total of 110 billion dollars economic loss and an average cost of $197 per victim [15]. To deal with the thousands of new

☆ This article is an extended/revised version of an ISCISC'12 paper.
* Corresponding author.
Email addresses: ghiasi@cse.shirazu.ac.ir (M. Ghiasi), asami@ieee.org (A. Sami), zsalehi@cse.shirazu.ac.ir (Z. Salehi)
ISSN: 2008-2045 © 2013 ISC. All rights reserved.

malware threats that are discovered every day, security companies and analysts use different techniques.

In the last few years, malware writers used code transformation methods such as encryption and obfuscation techniques to create different malware variants of the same family. So that most current detection methods are not able to detect them. Therefore, current malware detectors should be equipped with significant improvements to deal with new variants of malware.

The basic categorization of different approaches in malware detection consists of Static Analysis and Dynamic Analysis. Static analysis can be used to explore the context and the structure of a program with malicious intent without running it. Since tracing all execution paths of software is simple in static analysis, it has a low false positive rate. The static approach might not correctly analyze unseen new obfuscated malware [16–18].

Detection based on creating malicious signatures is a usual method that uses static analysis. Therefore, most commercial scanners maintain a database of signatures where a signature is an opcode pattern of the executable file that specifies the malware. By detecting a sequence of the binary code instructions that matches one of the known signatures in the database, the file will be classified as a member of the corresponding malware family. The accuracy of the system depends on the contents of the signature database, hence it should be updated permanently. Maintaining millions of signatures and delivering all updated signatures to users is severely time-consuming and costly. Besides, a new malware that contains a new signature might not be detected [19, 20].

Dynamic analysis executes malicious files in a simulated environment to recode their functionalities. Using dynamic analysis and monitoring the binarys operations performed, a profile of runtime behaviors will be collected. The major advantage of dynamic analysis is the ability of representing the actual intent of the program. This approach can expose the most precise malicious behaviors without static analysis of the code. On the other hand, packed malware automatically runs binary codes every time and the obfuscated code does not affect the final behavioral information. In addition, since the actual behavior is observed, it is not needed to estimate memory content, registers values or the lik [21].

In spite of the obvious advantages of dynamic analysis in detecting modern malware files, it has some drawbacks. Since dynamic malware analysis provides a limited aspect of malware behaviors, it is impossible to explore all execution paths and observing all variable values is not feasible. Some malware files only act in some specific conditions, hence dynamic analysis is unable to recognize their all behavior forms [18, 22]. For example, a bot carries out its malicious intent when its botmaster sends related commands through a command and control channel [23].

Obfuscation is the main reason for the better performance of dynamic analysis in comparison to static analysis. As mentioned above, static analysis alone cannot be adequate enough to detect malicious behaviors. Dynamic analysis is introduced as a complementary step besides static techniques because obfuscation has minor effects on it. Increasingly, more researchers follow dynamic analysis techniques to improve the effectiveness and accuracy of malware detection and classification [24]. For example, in [25] after logging run time behaviors, researchers used feature sets that are constructed based on APIs and/or input arguments to model the behaviors. [26] extracted unit string features from API call sequences, [27] used statistical properties of address pointers and size parameters and the temporal feature represents the order of invoked run-time API and [28] extracted the features from log files based on distinct string frequencies. DyVSoR also pursues the dynamic approach to identify malware files.

The main contribution of DyVSoR is to extend the Value Set Analysis (VSA) [29] that was proposed to detect new variants of a metamorphic malware. VSA is a static method that traces the distribution of values throughout an execution process. Based on the VSA, memory contents for different variants of polymorphic or metamorphic malware files almost remain unchanged. VSA was assumed not to be extendable to the dynamic technique because of the large number of existing values, but DyVSoR is extended to a dynamic setting.

Since the reports of our homemade tool contain all registers values of before and after API call invocations in each binary, the distributions and changes of registers can be traced throughout a binary. In addition, it is not required to approximate values by DyVSoR because dynamic analysis provides concrete run-time values.

A similarity function is used to describe the pairwise distance between each pair of malware applications or benign codes; and detection is simply done by the nearest neighbor search. Experiments show that this method is successful in identifying samples of diverse datasets with a low false positive rate. The preliminary work of DyVSoR, presented above, was based on 1211 samples of benign and malware files and DyVSoR was based on 1550 samples.

The rest of the paper is as follows. The related works that have used static and dynamic methods for identifying malware samples are explained in section 2. Next, an overview of DyVSoR system, the process of collecting tracing reports in the controlled environment and the process of matching phases for obtaining the similarity distance between all binary files are described in section 3. In section 4, the used dataset, the selection of common DLLs for most malware binaries and the measurement criteria are described. Experimental evaluations of the proposed method are given in section 5 and finally, the paper will be concluded and summarized followed by some future possible implications.

2 Related Works

A number of works have explored static and dynamic approaches to analyze and classify malware samples. These works are mentioned based on their methodologies. First, some static works are explored.

Some recent researches tend to use static analysis to distinguish clean files from malicious ones [30–36]. For example, Tian *et al.* [34] applied printable string information from the disassembly tool IDA as a feature to develop a malware categorization system. Walenstein *et al.* utilized PE header data, body information and their combination to detect unknown files. Tian *et al.*, and Walenstein *et al.* checked for the presence or absence of each feature and used several well-known classification algorithms using data mining techniques to classify the files. Sami *et al.* also [36] used the PE header to describe the behavior of executable files. They extracted the closed frequent patterns as the features. Ye *et al.* [35] used the analysis of static API call sequences as a feature. Sami *et al.*, and Ye *et al.* used the association rule mining besides the classification algorithms in order to detect the malware. The method of [36] improved state of the art technology in accuracy and false positive rate. Tahan *et al.* [37] also extracted 4-gram byte codes as features from files and constructed several classifiers to detect malware programs.

Leder *et al.*, [29] and Shankarapani *et al.* [38] presented a static approach to detect and classify metamorphic malware files by measuring the existing similarity between programs. Leder [29] measured the similarity between sets of values using Jaccard measure. Their approach identified the variants by tracing the use of consistent values throughout the malware. This attempt has led us to consider the contribution of malware detection based on registers values using Jaccard distance. Shankarapani [38] extracted API sequences that frequently appear in a number of malicious and applied Cosine measure, extended Jaccard measure, and the Pearson Correlation measures that are the popular measures of similarity for sequences.

As already mentioned anti-antivirus techniques, such as polymorphism and metamorphism, make it difficult to detect malware using static extraction alone. Therefore, dynamic analysis is needed as a complement for static techniques [39].

Dynamic analysis is applied to overcome the limitations of signature-based detection. Some researchers used API calls and their related properties to model the behavior of samples through a graph. They built their graph in different ways and analyzed and compared graphs using different methods.

Christodorescu *et al.* [40] monitored a specification of malicious intent that the malware does to affect the host operating system. The extracted and compared the system call dependency graphs of malware and benign applications to detect malicious behaviors. Bai *et al.*, [41] and Guo *et al.* [42] applied behavior Control Flow Graph (CFG) and then used critical API Graph based on CFG to match the sub-graphs. Park *et al.*, [43] and Karbalee *et al.* [44] proposed a new malware classification method based on maximal common sub graph detection. They considered each system call as a graph node, the system call sequence as the edges and each dependency of the two system calls as a label for edges. [44] is the first work that used frequent graph mining to detect an unseen program. Sub-graph mining produced more accurate algorithms in comparison to exact graph matching. Kostakis *et al.* [45] created graphs from the subroutines as nodes and their call references as edges. Park and Reeves [46] used kernel objects as node graphs and Elhadi *et al.* [16] built it from the API calls and the operating system resources used by API call.

The above listed works may contain a huge number of nodes and edges and this number needs to be minimized. On the other hand, comparing graphs to find the existing similarities between them is time and space consuming because some problems are NP-complete [16].

What follows is an account of the recent dynamic works in which researchers used data mining and machine learning techniques to classify the programs. Ahmed *et al.* [27] combined spatial and temporal features in run-time API calls to build a model and distinguish between benign and malware files. Tian *et al.* [28] considered each API Call and other extracted features as string information to represent the program behavior. They transformed a report into a vector that included these string features and then passed it on to the classification phase. Zhao *et al.* [26] extracted unit string features from those trace reports which included the changed status caused by the executables and the event which was transferred from the corresponding Win32 API calls and their certain parameters. Ahmadi *et al.* [47] used iterative system call pattern mining to detect malware. They believed that repetitive actions on data sequences are often used by malware writers, especially some famous loops performing decryption or encryption and infection. Salehi *et al.* [25] assumed APIs cannot represent the similar behavior of samples properly; therefore they extracted API calls and input arguments together as predictive features.

Some works used the similarity measures between samples to detect malware files. Wegenger *et al.* [48] compared two cases of malware behavior with each other by observing all the system function calls, and leveraged the Hellinger distance to compute the associated distances. Bayer *et al.* [49] proposed an approach for clustering malware samples so that the malware files with similar behavior could be grouped together. They extracted the profiles of binary behaviors such as system calls, the dependencies between system calls and network analyses. They used the Jaccard Index to measure the similarities between executable files. Since the huge number of new malicious files are mutated samples of only a few malware instances, Bayer *et al.* [50] proposed a system to avoid analyzing mutated instances of malware binaries that are already analyzed. They dynamically analyzed each malware file for a short period of time to realize if it is a mutated instance of an already analyzed binary. Finally, they calculated a pair-wised similarity between all the behavioral trace reports. Jang *et al.* [51] presented a system that clustered malware based on Jaccard similarity distance. They believed that malware files that share large features are more similar. They evaluated their system using two analysis methods; static and dynamic. First, in static analysis, two files are considered as similar if they use large fragments of the same code as features. In dynamic analysis, two files are considered similar if they represent similar behavior. Hegedus *et al.* [52] also used Jaccard measure and K-nearest neighbor classifier to predict whether an unknown executable is malicious or benign using behavioral data.

As mentioned above in the introduction, VSA is the base of this research, but Leder [29] used a small dataset and the estimation of memory locations is very complicated. What makes the present experiment different from the VSA method is using Dynamic analysis and the diversity of the used dataset. Besides, this proposed system eased the estimation of memory locations in comparison to the complicated memory estimation in VSA.

3 DyVSoR Malware Detection System

In this article, we discuss a system that is used to detect malicious files. The proposed system analyzes the behavior of executables in the run-time mode. Figure 1 depicts the basic analysis steps of the proposed system. In the first step, all files are executed in a controlled environment to record their actions. The recorded actions are stored in XML format and are called trace reports. In the next step, the registers contents of each trace report are used to extract an analysis of the behavioral patterns. The matching phase calculates the similarity distance between all the executables based on the registers contents to distinguish between the malicious cases of behavior and the clean ones.

The process of monitoring the behavior of executables is described in section 3.1. Then Section 3.2 addresses the main contribution of using the value sets of registers.

3.1 Monitoring the Run-Time Behavior of Binaries in a Controlled Environment

To implement DyVSoR, the behaviors of binaries are logged using a tool which is developed by our team to provide a controlled environment. The reports of the binary file executions are collected based on the API system calls. A homemade tool which includes several parts is produced: VMWare V.8.0.0-471780 is installed on the host OS, Windows XP. After installing the virtual machine, a hooking tool based on [53] is installed on it and the binaries are run under that tool.

This homemade tool could capture all the invoked API calls by a binary along with all its arguments, registers contents before and after API calls, API returned value, etc. The tool injects a library for intercepting the arbitrary Win32 binary functions on x86 machines. The interception code is applied dynamically at the run-time. The tool attaches arbitrary DLLs to the target process, then the DLLs initialize interprocess communication between the hooking tool and the function calls that are hooked. This directs the tool to monitor the desired APIs. This tool can break a targeted application before or after an invoked function call and capture and return its registers values and input arguments.

To start the process of dynamic analysis, a clean snapshot is taken of the state of VMWare. A binary is run under the mentioned homemade tool for two minutes. This time is selected because a time of two minutes is generally found to be enough for most malware files to execute their immediate payload; if they have one [54]. The developed controlled environment is shown in Figure 2. Finally the trace reports of the binary files are collected; VMWare is reverted to the clean snapshot after running each file. This step is performed to prepare a clean environment for executing the next files and prevent the behavior of other executed binaries from being affected by the already executed binaries. An output trace report is shown in Figure 3.

The Static VSA tries to specify the values for all the memory contents at each line of code and it has to approximate all the values that an instructions operand

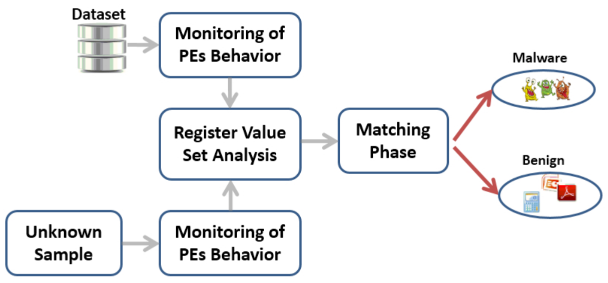

Dataset
Monitoring of PEs Behavior
Register Value Set Analysis
Matching Phase
Malware
Benign
Unknown Sample
Monitoring of PEs Behavior

Figure 1. The architecture of DyVSoR system

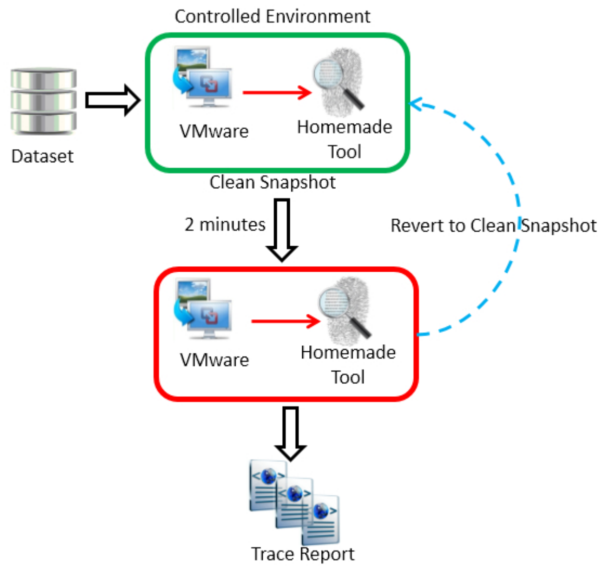
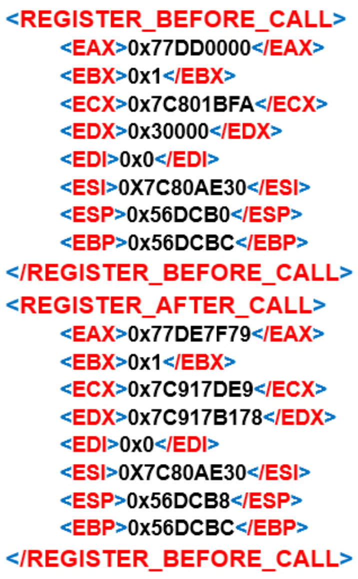


Figure 2. Running sample files in the controlled environment



```
<REGISTER_BEFORE_CALL>
    <EAX>0x77DD0000</EAX>
    <EBX>0x1</EBX>
    <ECX>0x7C801BFA</ECX>
    <EDX>0x30000</EDX>
    <EDI>0x0</EDI>
    <ESI>0X7C80AE30</ESI>
    <ESP>0x56DCB0</ESP>
    <EBP>0x56DCBC</EBP>
</REGISTER_BEFORE_CALL>
<REGISTER_AFTER_CALL>
    <EAX>0x77DE7F79</EAX>
    <EBX>0x1</EBX>
    <ECX>0x7C917DE9</ECX>
    <EDX>0x7C917B178</EDX>
    <EDI>0x0</EDI>
    <ESI>0X7C80AE30</ESI>
    <ESP>0x56DCB8</ESP>
    <EBP>0x56DCBC</EBP>
</REGISTER_BEFORE_CALL>
```

Figure 3. A portion of an output trace report from the developed tool

may contain. But DyVSoR system is not complicated in estimating memory locations in comparison to that in static VSA. The Dynamic VSA assigns the run-time and also concrete values to each memory location at a point in the run-time. It aims at collecting the data without any human intervention to approximate the content.

Also, since the homemade tool reports the registers before and after invoking API calls for each binary and is unable to capture the stack and memory contents, the propagation and changes of registers values are traced throughout the process.

3.2 Value Sets of Registers

This section addresses the identification of malware binaries by using registers contents. The value set analysis approach is briefly described in section 3.2.1 What follows is a brief introduction to the procedure of matching phase.

3.2.1 Value Set Analysis (VSA)

VSA is a static method that tracks the propagation and variation of values throughout an executable binary without executing it. VSA estimates the memory content for all the instructions of the executable binaries to detect and classify polymorphic and metamorphic samples. The memory content includes registers contents, global memory, stack and heap allocations. Based on the VSA idea, memory contents for different variants of some polymorphic or metamorphic malware remain mostly unchanged. Since this malware alters its appearance indifferent ways, such as by inserting some junk code, instruction reordering and variable renaming, the behavior of the underlying malicious code body does not change. Therefore, the malicious functionality will not change and the behavior can be reflected by memory contents.

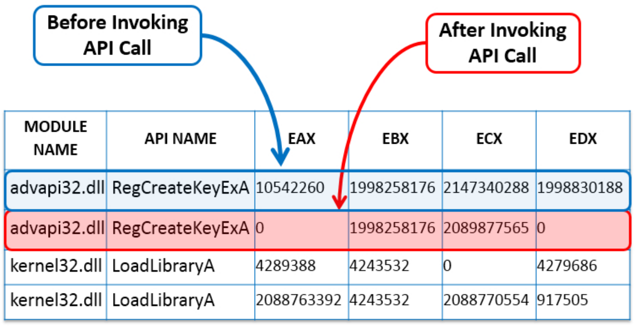
In general, VSA uses value and memory contents to identify malware files. The example presented in Figure 4 describes the VSA idea more clearly. In VSA method, if the similarity score between the two files $File_A$ and $File_B$, reaches a specified threshold then $File_A$ and $File_B$ are two variants of a polymorphic malware. Based on VSA approach, they achieved a 100% detection rate on 25 benign files and 25 malware files from six families. Obviously, the sample size is too small to generalize the idea to large malware and benign population. VSA was assumed not to be extendable to a dynamic setting because of the large number of values.

3.2.2 Matching Phase based on RVSA

In the matching phase based on DyVSoR, the run-time registers values of binaries are used to obtain

	Value Sets
mov EAX, 4	EAX = {4};
mov EBX, 3	EBX = {3};
mov EBX, EAX	EAX = {4, 3}; EBX = {3}
add EAX, EBX	EAX = {4, 3, 7, 6}; EBX = {3}
sub EAX, EAX	EAX = {4, 3, 7, 6, 0}; EBX = {3}

Figure 4. VSA idea


Before Invoking API Call
After Invoking API Call

MODULE NAME	API NAME	EAX	EBX	ECX	EDX
advapi32.dll	RegCreateKeyExA	10542260	1998258176	2147340288	1998830188
advapi32.dll	RegCreateKeyExA	0	1998258176	2089877565	0
kernel32.dll	LoadLibraryA	4289388	4243532	0	4279686
kernel32.dll	LoadLibraryA	2088763392	4243532	2088770554	917505

Figure 5. The new representation of a trace report

the similarity scores between all binary files. To ease this process, the run-time trace report L, as shown in Figure 3 is transformed to another format. The trace report L is converted to the matrix M. Each row of the matrix M belongs to one of the invoked API calls in the trace report L. In each row, the API call is shown with all its features. The new representation of the trace reports is illustrated in Figure 5.

The goal of the matching phase is to compute the similarities between all binary files. Suppose that the dataset contains ten files: $File_1$, $File_2$, ... and $File_{10}$, to obtain similarities ($File_1$, $File_2$), ..., ($File_1$, $File_{10}$), ($File_2$, $File_3$), ..., ($File_2$, $File_{10}$), ..., ($File_9$, $File_{10}$) should be computed separately. For example, to carry out the dynamic VSA on $File_1$ and $File_2$, the dynamic VSA should be performed on all the registers of $File_1$ and $File_2$; that is the similarity scores between (EAX_1, EAX_2), (EBX_1, EBX_2), (ECX_1, ECX_2), ...should be computed. The similarity scores between all the registers of both $File_1$ and $File_2$, are then arranged in matrix M_{12}.

$File_1$	$File_2$	(EAX_1, EAX_2)	(EBX_1, EBX_2)	...

To calculate the similarity score between EAX_1 and EAX_2, first the two vectors V_1 and V_2 are generated for the values of EAX_1 and EAX_2 respectively. The Jaccard distance formula is defined as equation (1).

$$J_{EAX} = \frac{|V_1 \cap V_2|}{|V_1 \cup V_2|} \tag{1}$$

Test File Name	Second File Name	Type of Second File (malware/benign)	Similarity score
$File_1$	$File_2$	1/0	J (1,2)
$File_1$	$File_3$	1/0	J (1,3)
$File_1$	....	....	....
$File_1$	$File_{\lvert Test \rvert}$	1/0	J (1, |Test|)
$File_2$	$File_3$	1/0	J (2,3)
$File_2$	....	....	....
$File_2$	$File_{\lvert Test \rvert}$	1/0	J (2, |Test|)
...	...	...	...
$File_{\lvert Dataset \rvert -1}$	$File_{\lvert Test \rvert}$	1/0	J(|Dataset|-1, |Test|)

Figure 6. Similarity score Matrix M

Jaccard similarity is a statistic measure that is used for comparing the similarity of sample sets. The numerator calculates the amount of the shared features between the two samples V_1 and V_2, and the denominator is the size of the union of the sample sets. The reason behind using this formula is that the degree of similarity is relative to the percentage of the shared features, e.g., the malware files that share most of the same features are more similar than the ones that do not.

Equation (1) is repeated for (EBX_1, EBX_2), (ECX_1, ECX_2), ... The effect of just one register in detection was examined regardless of other registers and the similarity between the corresponding registers of the two files can be calculated. The previous steps are repeated for all files. The pair-wise Jaccard distance between all binaries is calculated and the similarity score matrix M is generated which is observable in 6.

The similarity score matrix M is used to identify and classify the test file into either the malware or the benign class. Hence, the highest similarity score is obtained between each test file and other files in the dataset. The test file is assigned to the file class with which it has the highest similarity.

The obtained similarity value is considered as the similarity score to explore the impact of the registers on malware detection. The code in Figure 7 shows the process of matching phase if the similarity between the EAX registers of the two files was taken as the similarity score. This algorithm has the order of | *Dataset* | time complexity because to identify the nature of a test file, the similarity distance between the unknown file and each file in the dataset should be computed.

```
WHILE  i <= |Dataset|
        $HM_i = J(EAX_i, EAX_{Test})$
      IF  High-Similarity < $HM_i$
          Assign $HM_i$ to High-Similarity
          Assign Lable-File$_i$ to Label-File$_{Test}$
      ENDIF
ENDWHILE
```

Figure 7. The process of matching phase

Table 1. Experimental dataset

Type of Family	Number of files
Backdoor	200
P2P-Worm	200
Trojan	200
Worm	200
Virus	200
Benign	550
Total Sum	**1550**

4 Data and Experimental Setting

The dataset used in this research includes malware and benign files. The variety of malware and benign PEs in the dataset is described in 4.1. The selection process of six well-known and common DLLs for most malware binaries is explained in 4.2. The performance of the proposed system is assessed by some evaluation measures. These measures are briefly explained in part 4.3.

4.1 Data Collection

The used dataset in the experiments [36] includes 550 benign files and 1000 malware samples that can be categorized in five groups: Backdoors, P2P-Worms, Trojans, Worms and Viruses. The distribution of the used dataset is illustrated in Table 1. The detailed list of the examined executables can be obtained at the following URL: `http://home.shirazu.ac.ir/~sami/malware`. The diversity of the dataset is another difference between this approach and the VSA approach. The larger number of samples aims at having reliable results.

4.2 Essential API Calls for Detecting Malicious Behavior

Although in this article, all API calls are not monitored, but a subset of 126 API calls from six important DLLs is logged. The mentioned DLLs are as follows: advapi32.dll, kernel32.dll, ntdll.dll, user32.dll, wininet.dll and ws2_32.dll . These DLLs are chosen by considering the recommendations of the previously done researches by our group like Karbalaee *et al.* [44].

Karbalaee *et al.* [44] used data mining algorithms to find a subset of system calls by these DLLs that are important for malicious activity detection. Four hundred malware and 397 benign applications ran all the mentioned six DLLs and all the API calls were monitored to diagnose which system call is more important in malware detection. They ran each malware and benign program for 30 seconds. All system calls that each malware and benign file invoked were collected. Afterwards, 10-fold cross validations [55] with random forest classifiers [56]were used to measure the accuracy rate of the selected system calls. Then some feature selection techniques have been used to select the most discriminative system calls.

They concluded that it is essential to consider only 126 system calls that are related to six DLLs to be used in analyzing malware behavior. These DLLs are commonly recognized for most malware files, as briefly described in Table 2. Some APIs of these DLLs are given in Table 3.

Table 2. **A summary of the selected DLLs**

DLL Name	Description
Kernel32.dll	Performing Low-level operating system functions such as memory management, input/output operations, process and thread creation, and synchronization functions [57].
User32.dll	Windows user component management such as creating and manipulating standard elements of the windows user interface, desktop, windows, and menus [57].
Advapi32.dll	Accessing additional functionalities of the kernel such as windows registry, system shutdown/restart (or abort), start/stop/create a windows service and managing user accounts [57].
Ntdll.dll	Exporting the Windows Native API [57].
Wininet.dll	Protocol handler for HTTP, HTTPS and FTP [57].
WS2-32.dll	Contains Windows Sockets API used by most internet and network applications to handle network connections [57].

Table 3. **Some API common DLLs to detect malware**

DLL Name	API Name	DLL Name	API Name
advapi32.dll	RegLoadKeyA	kernel32.dll	CreateProcessA
advapi32.dll	RegLoadKeyW	kernel32.dll	CreateProcessW
advapi32.dll	RegOpenKeyA	kernel32.dll	DeleteFileA
advapi32.dll	RegOpenKeyExA	kernel32.dll	DeleteFileW
...	...	...	...
ntdll.dll	NtSaveKey	user32.dll	LoadImageA
ntdll.dll	NtSetValueKey	user32.dll	CreateWindowExA
ntdll.dll	NtUnloadKey	user32.dll	GetDlgItem
ntdll.dll	SetWindowTextW	user32.dll	GetFocus
...	...	...	...

4.3 Performance Evaluation Criteria

The performance of DyVSoR system is measured with respect to four metrics: Recall, Precision, F-Measure and Accuracy. Recall is the ratio of the malware programs correctly predicted from the total number of malicious programs, Precision is the ratio of the malware samples correctly identified from the total number of programs identified as malware. Precision and recall are defined as equation (2) and (3) respectively:

$$Precison = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

True Positive (TP) represents the correctly identified malware samples. False Positive (FP) represents the incorrectly classified benign samples. True Negative (TN) represents the correctly identified benign samples and False Negative (FN) represents the incorrectly identified malware samples. F-Measure is the harmonic mean of precision and recall. The F-Measure formula is defined as (4).

$$F - Measure = \frac{2 * Precison * Recall}{Precison + Recall} \tag{4}$$

Accuracy is also used as a performance measure of how well the model correctly identifies malware samples. The formula of accuracy which is the proportion of true results (both true positives and true negatives) in the dataset is shown in (5).

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5}$$

Accuracy alone is only appropriate for data mining experiments with balanced datasets. That is, the numbers of malware and benign samples are almost equal but here, the used dataset is imbalanced because the dataset contains just about 30% benign applications. It is shown that precision, recall and F-Measure are good measures for imbalanced datasets [58]. In this paper, F-Measure and false positive are used as measures to evaluate the malware detection rate of the proposed system.

It is noteworthy that in the present experiments, k-fold cross validations were used to classify the malware and benign files, where k = 10. This is a well-known standard of classification in a number of domains including malware analysis [28]. Malware and benign files are selected and are divided into 10 folds randomly. In each experiment, nine folds are chosen for the training dataset and the last one is considered for the test dataset. The model created based on the training dataset can be evaluated by examining the results of classifying the test data. The process is repeated for all the 10 folds. The process of 10-fold cross validation is represented in 8.

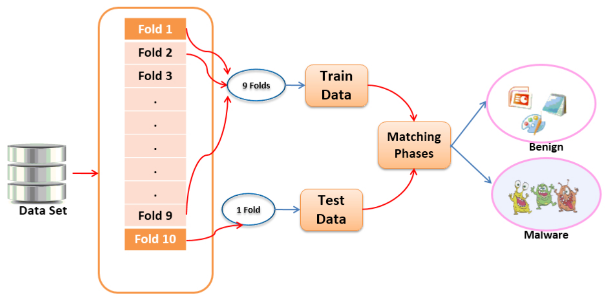


Figure 8. Ten-fold cross validation procedure

5 Experimental Evaluation

Firstly, to generate the log files, each binary of the dataset was run in the homemade tool and all the mentioned values were recorded. Log files were generated based on the API names, registers values, input arguments and returned values. As stated above, all experiments were evaluated using 10-fold cross validations to avoid over-fitting. In a 10-fold cross validation, datasets are randomly divided into 10 equal parts while keeping the same class distribution as the original file in each part. Each experiment was run 10 times and each time 9 parts of the data were used as the training data and the remaining part was then used as the test data.

The distribution and propagation values of registers for detecting malware samples were then explored. In each experiment, the effect of just one register in detection was examined regardless of the other registers. In other words, in each experiment we only had one register. Table 4 shows the detection rate of all registers. As observable in 4, the values of EAX register have a major distinguishing effect on identifying the malware samples and were able to categorize files with an F-Measure of 96% and a false positive rate of 4%. The EAX register is a 32-bit general-purpose integer register and is the accumulator one. It is used for I/O port access, arithmetic operations, interrupting calls, logic, and data transfer, multiplication, division, etc [59]. Also the return value of a function is located in EAX in 4 bytes or less [60]. If the length of the returned value exceeds four bytes, the combination of EDX, stack, and EAX is used to provide the function returned value. In this experiment, the similarity scores between the EAX registers were assumed equal to the similarity scores between the files.

Intel 80x86 ABI [61] specifies that EAX is a scratch register and Win32 functions follow Intel ABI. Most Win32 API functions return a return-value in EAX and most calculations occur in the accumulator. As a result, the architecture of x86 contains many optimized instructions to move the data in and out of this register, which is why this experiment works so well.

In the next experiment, the impact of EBX register on detecting malware samples is examined. EBX, which is called the base register, acts as a general-purpose pointer. It is used as a base pointer for memory access.

The performance of ECX and that of EDX are explored in the next two experiments. The data register EDX is used for I/O port access, arithmetic operations and some interrupting calls. EDX is very similar to the accumulator. The counter register ECX is used as a loop counter. It is also used for shifts.

Table 4 also shows the results of EDX and ECX on dataset samples. These experimental evaluations demonstrate that EDX is successful in identifying the samples of the diverse dataset by 94.8% and ECX identifies them with an F-measure of 93.8%.

EDI, the destination index, holds the implied write address of all string operations. Each loop that generates data must store the results in memory, and it requires a moving pointer and EDI is this pointer. EDI is successful in identifying 93.2% of the samples with a false positive rate of 8.5%. The source index, ESI, has the same properties as the destination index. The only difference is that the source index is for reading rather than writing. ESI identifies 94.4% of the samples. The base pointer (EBP) holds the location of the current stack frame in functions that store parameters or variables on the stack [59]. The Intel 80x86 ABI specifies that functions must preserve the values of certain EBX, ESI, EDI, and EBP registers across a function call. If the function needs to modify the value of any of those registers, it must save those registers values and restore them before returning to the caller [59].

The experimental results based on a dataset of 1000 malware and 550 benign files provide an average accuracy of over 94.5% in distinguishing malware applications from benign ones. In the best case, it detected 96% malware samples with a false positive rate of 4%.

Since the related experiments have used different datasets and sandboxes, the outputs might have various formats, hence its impossible to make a comparison among them and the present work [37].

In order to examine the efficiency of the registers sets, the same dataset as the ones described in section 4.1 are used to make a comparison with the outputs of the other anti-virus tools. Our results out performed some updated common anti-virus applications. Table 5 compares the efficiency of different antivirus scanners on the dataset.

Table 4. Matching results by registers

Register Name	TP	FP	Precision	Recall	F-M	ACC
EAX	0.961	0.04	0.961	0.96	0.96	0.961
EBX	0.945	0.055	0.945	0.945	0.945	0.945
ECX	0.946	0.07	0.946	0.931	0.938	0.938
EDX	0.958	0.062	0.958	0.939	0.948	0.948
ESI	0.957	0.07	0.957	0.932	0.944	0.944
EDI	0.947	0.085	0.947	0.918	0.932	0.931
EBP	0.953	0.06	0.953	0.941	0.947	0.947
Mean	0.952	0.063	0.952	0.938	0.945	0.945

Table 5. Result of system in comparison to some of the updated antiviruses

	ESET Node32	KasperSky	Avira	DyVSoR
Detection Rate	89.3%	92.6%	98.4%	96%

6 Future Work and Conclusion

In this paper the behavioral information is extracted from both malware and benign files to distinguish malware files from benign ones. A dynamic analysis of registers values in the event of API calls was performed. This paper is the first work that deploys register values in dynamic malware detection. The assumption of the paper is based on the fact that the run-time behavior of the underlying malicious code body could be expressed in registers contents. The impact of some registers on identifying and classifying malware samples was demonstrated.

The proposed method was evaluated on a large dataset of malware and benign files. The empirical results demonstrated the efficacy of the proposed method in correctly distinguishing 96.1% of the malware from the benign with a false positive rate of 4%.

In future studies, this method will be extended to an incremental approach for behavior-based analysis. This method will be used to categorize malware families and types.

Acknowledgment

We would like to thank the referees and the editor for a number of helpful comments and suggestions. This research was conducted in the Department of Electrical Engineering and Computer Science of Shiraz University and we would like to express our sincere thanks to **Iran Telecommunication Research Center (ITRC)** for their supports.

References

[1] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-Aware Malware Detection," IEEE Symposium on Security and Privacy (S&P05), Washington. DC. USA, pp. 32-46, 2005.

[2] Symantec Corp, "Symantec Global Internet Security Threat Report," Vol. 7, 2008.

[3] PandaLabs, "Pandalabs annual malware report 2009," 2010.

[4] K. Kim, and B. R. Moon, "Malware detection based on dependency graph using hybrid genetic algorithm," Proceedings of the 12th Annual Conf. on Genetic and Evolutionary Computation, ACM. USA., pp. 1211-1218, July 2010.

[5] McAfee Labs, "McAfee Threats Report: Fourth Quarter 2010," McAfee Inc., Santa Clara. California, 2010.

[6] X. Hu, "Large-Scale Malware Analysis, Detection, and Signature Generation," A dissertation for the degree of Doctor of Philosophy, University of Michigan, Ann Arbor. Michigan. United States, 2011.

[7] P. Wood, M. Nisbet, G. Egan, N. Johnston, K. Haley, B. Krishnappa, T. K. Tran, I. Asrar, O. Cox, S. Hittel, *et al.*, "Symantec Internet Security Threat Report Trends for 2011," Vol. 17, Symantec Corporation, 2012.

[8] PandaLabs, "Pandalabs annual malware report 2011," 2012.

[9] Panda Security, "PandaLabs Annual Report 2012," 2013.

[10] Sophos, "Security threat report 2013 New Platforms and Changing Threats," Sophos Ltd., Boston, USA, 2013.

[11] Macafee Labs, "McAfee Threats Report: Fourth Quarter 2012," McAfee Inc, 2013.

[12] Symantec Corporation, "Internet Security Threat Report 2013," Vol. 18, 2013.

[13] Sophos, "Security threat report 2011," Sophos Ltd., Boston, USA, January 2011.

[14] Symantec Corporation, "The Shamoon Attacks," [On-line]. Available electronically at *http://www.symantec.com/connect/blogs/shamoon-attacks*. 2012.

[15] Norton by Symantec, "2012 Norton Cybercrime Report," 2012.

[16] A. E. Ammar, A. M. Mohd, and H. Ahmed, "Malware Detection Based on Hybrid Signature Behaviour Application Programming Interface Call Graph," American J of Applied Sciences, United States, vol. 3, pp 283-288, 2012.

[17] L. Bohne, "Pandoras Bochs: Automatic Unpacking of Malware," Diploma Thesis, University of Mannheim, January 2008.

[18] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware analysis techniques and tools," ACM Computing Surveys (CSUR) J., Vol. 44, ACM. New York. USA, pp. 1-49, February 2012.

[19] S. M. Abdulalla, L. M. Kiah, and O. Zakariam, "A biological Model to Improve PE Malware Detection: Review," Int. J. of Physical Sciences, vol. 5, pp. 2236-2247, 2010.

[20] K. M. Goertzel, "Tools on Anti Malware," Technical Information Center, 2009.

[21] Li. Shengying, "A survey on tools for binary code analysis," Stony Brook University, August 2004.

[22] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J. Nazario, "Automated Classification and Analysis of Internet Malware," In Proceedings of Symposium on Recent Advances in Intrusion Detection (RAID07), pp. 178-197, 2007.

[23] M. Yahyazadeh, and M. Abadi, "BotOnus: An Online Unsupervised Method for Botnet Detection," The ISC Int. J. of Information Security (ISeCure), vol. 4, pp. 51-62, January 2012.

[24] P. Li, L. Liu, D. Gao, and M. K. Reiter, "On challenges in evaluating malware clustering," In Proceedings of the 13th Int. Conf. on Recent advances in intrusion detection (RAID10), Berlin. Heidelberg, pp. 238-255, 2010.

[25] Z. Salehi, M. Ghiasi, and A. Sami, "Malware Detection Preserving API Function Calls and Their Standard Function Calling Notation," In Proceeding of 16th CSI Symposium on Artificial Intelligence and Signal Processing (AISP 2012), Shiraz, Iran, 2012.

[26] H. Zhao, M. Xu, N. Zheng, J. Yao, and Q. Ho, "Malicious executables classification based on behavioral factor analysis," In Proceeding Int. Conf. on e-Education, e-Business, e-Management and e-Learning (IC4E 2010), Sanya. China, pp. 502-506, 2010.

[27] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq, "Using spatiotemporal information in api calls with machine learning algorithms for malware detection," In Proceeding Second ACM workshop on Security and artificial intelligence (AISec 09), New York, USA, pp. 55-62, 2009.

[28] R. Tian, R. Islam, and L. Batten, "Differentiating Malware from Cleanware Using Behavioral Analysis," In Proceeding Fifth Int. Conf. on Malicious and Unwanted Soft-ware (MALWARE 2010), Nancy, France, pp. 23-30, 2010.

[29] F. Leder, B. Steinbock, and P. Martini, "Classification and detection of metamorphic malware using value set analysis," In Proceeding Fourth Int. Conf on Malicious and Unwanted Software

(MALWARE 2009), pp. 39-46, 2009.

[30] V. S. Sathyanarayan, P. Kohli, and B. Bruhadeshwar, “Signature Generation and Detection of Malware Families,” In Information Security and Privacy 13th Australasian Conf. (ACISP 2008),Wollongong, Australia, pp. 336-349, July 2008.

[31] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, and Y. Elovici, “Unknown Malcode Detection via Text Categorization and the Imbalance Problem,” Intelligence and Security Informatics (ISI 2008), Taipei. Taiwan, pp. 156-181, 2008.

[32] I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas, “Idea: Opcode-sequence-based malware detection,” In Engineering Secure Software and Systems Second Int. Symposium (ESSoS 2010), Pisa. Italy, pp. 35-43, February 2010.

[33] R. Tian, L. M. Batten, and S. C. Versteeg, “Function Length as a Tool for Malware Classification,” In Proceedings of the 3rd Int. Conf. on Malicious and Unwanted Software (Malware 2008), pp. 69-76, 2008.

[34] R. Tian, L. Batten, R. Islam, and S. Versteeg, “an Automated Classification System based on the Strings of Trojan and Virus Families,” In Proceedings of the 4th Int. Conf. on Malicious and Unwanted Software (MALWARE 2009), Quebec. Canada, pp. 23-30, October 2009.

[35] Y. Ye, T. Li, Q. Jiang, and Y. Wang, “CIMDS: Adapting Post processing Techniques of Associative Classification for Malware Detection,” IEEE Trans. Systems, Man, and Cybernetics, Part C: Applications and Reviews, Vol. 40, pp. 298-307, May 2010.

[36] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, “Malware detection based on mining API calls,” In Proceedings of ACM Symposium on Applied Computing (SAC 10), Switzerland, pp. 1020-1025, March 2010.

[37] G. Tahan, L. Rokach, and Y. Shahar, “Mal-ID: Automatic Malware Detection Using Common Segment Analysis and Meta-Features,” The J. of Machine Learning Research, Vol. 13, pp. 949-979, 2012.

[38] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, and S. Mukkamala, “Malware detection using assembly and API call sequences,” J. in Computer Virology, Vol. 7, pp. 107-119, 2010.

[39] P. M. Comparetti, G. Salvaneschi, E. Kirda, C. Kolbitsch, C. Kruegel, and S. Zanero, “Identifying Dormant Functionality in Malware Programs,” IEEE Symposium on Security and Privacy (S&P 2010), Berleley/Oakland. California. USA, pp. 61-76, May 2010.

[40] M. Christodorescu, S. Jha, and C. Kruegel, “Mining specifications of malicious behavior,” Foundations of Software Engineering, pp. 1-10, 2007.

[41] L. Bai, J. Pang, Y. Zhang, W. Fu, and J. Zhu, “Detecting malicious behavior using critical API calling graph matching,” Proceedings of the 1st Int. Conf. on Information Science and Engineering, Nanjing, pp. 1716-1719, 2009.

[42] H. Guo, J. Pang, Y. Zhang, F. Yue, and R. Zhao, “HERO: A novel malware detection framework based on binary translation,” Proceedings of the IEEE Int. Conf. on Intelligent Computing and Intelligent Systems, Xiamen, pp. 411-415, 2010.

[43] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, “Fast malware classification by automated behavioral graph matching,” Proceedings of the 6th Annual Workshop on Cyber Security and Information Intelligence Research, USA, 2010.

[44] F. Karbalaee, A. Sami, and M. Ahmadi, “Semantic Malware Detection by Deploying Graph Mining,” Int. J. of Computer Science Issues (IJCSI 2012), Vol. 9, pp. 373-379, 2012.

[45] O. Kostakis, J. Kinable, H. Mahmoudi, and K. Mustonen, “Improved call graph comparison using simulated annealing,” Proceedings of the 2011 ACM Symposium on Applied Computing, USA, pp. 1516-1523, 2011.

[46] Y. Park, and D. Reeves, “Deriving common malware behavior through graph clustering”, Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, USA, pp. 497-502, 2011.

[47] M. Ahmadi, A. Sami, H. Rahimi, and B. Yadegari, “Iterative System Call Patterns Blow the Malware Cover,” IT Security for The Next Generation, Asia Pacific & MEA Cup 2011, Malaysia, March 2011.

[48] G. Wagener, R. State, and A. Dulaunoy, “Malware behaviour analysis,” J. in Computer Virology, Vol. 4, pp. 279-287, 2008.

[49] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, “Scalable, Behavior Based Malware Clustering,” Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS’09), San Diego, February 2009.

[50] U. Bayer, E. Kirda, and C. Kruegel, “Improving the Efficiency of Dynamic Malware Analysis,” In Proceedings of the 2010 ACM Symposium on Applied Computing (SAC ’10), NY, USA, pp. 1871-1878, 2010.

[51] J. Jang, D. Brumley, and S. Venkataraman, “BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis,” Proceedings of the 18th ACM conf. on Computer and Communications Security, ACM, pp. 309-320, 2011.

[52] J. Hegedus, Y. Miche, A. Ilin, and A. Lendasse,

"Methodology for Behavioral-based Malware Analysis and Detection Using Random Projections and K-Nearest Neighbors Classifiers," In Proceedings of the Seventh Int. Conf. on Computational Intelligence and Security, Sanya, Hainan, China, pp. 1016-1023, 2011.

[53] J. Potier, "WinAPIOverride32," 2013. [Online]. Available electronically at *http://jacquelin.potier.free.fr/winapioverride32/*.

[54] M. Fredrikson, S. Jha, M. Christodorescu, "Synthesizing Near-Optimal Malware Specification from Suspicious Behaviors," Proceeding 31st IEEE Symposium on Security and Privacy (S&P 2010), pp. 45-60, 2010.

[55] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," In Proceedings of the Fourteenth Int. Joint Conf. on Artificial Intelligence, pp. 1137-1143, 1995.

[56] L. Breiman, "Random Forests," Kluwer Academic Publishers. Manufactured in The Netherlands. 2001.

[57] T. Langerud, "PowerScan: A Framework for Dynamic Analysis and Anti-Virus Based Identification of Malware," Master thesis, Norwegian University of Science and Technology Department of Telematics, Nor-way, 2008.

[58] C. G. Weng, and J. Poon. "A New Evaluation Measure for Imbalanced Datasets," In Seventh Australasian Data Mining Conf. (AusDM 2008), pp. 27-32, 2008.

[59] A. Fog, "Function calling conventions," In Calling conventions for different C++ compilers and operating systems, Copenhagen, Denmark, 2012.

[60] J. Potier, "Where is located the return value?," [On-line]. Available electronically at *http://jacquelin.potier.free.fr/winapioverride32/doc/faq.htm#returnvalue*, 2011.

[61] Intel Corporation, "Intel Itanium Processor specific Application Binary Interface (ABI) Intel," 2001.

Mahboobe Ghiasi has obtained her B.S. degree in Computer Science in 2009 from Azad Shiraz University. Since 2010, she is a master student at Computer Engineering at Shiraz University. Worked on malware detection as her M.S. theses under supervision of Dr. Sami. Her research interests include security and data mining.

Dr. Ashkan Sami has obtained his B.S. from Virginia Tech; Blacksburg, VA; U.S.A., M.S. from Shiraz University; Iran and Ph.D. from Tohoku University; Japan. He is interested in data mining, software quality and security. Ashkan has been a member of technical committee of several international conferences like PAKDD, ADMA, HumanCon, and Future Tech, and has more than 40 conference paper and nearly 10 journal papers. He is an associate member of IEEE and was among the founding members of Shiraz University CERT.

Zahra Salehi has obtained her B.S. degree in Computer Science in 2009 from Azad Shiraz University. Since 2010, she is a master student at Computer Engineering at Shiraz University. She focused on malware detection during her M.S. studies under supervision of Dr. Sami. Her research interests include security and data mining.