A Survey on Heuristic Malware Detection Techniques

Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, Ali Hamzeh
Department of Computer Science and Engineering
Shiraz University
Shiraz Iran
{zbazrafshan, h-hashemi, hazrati, ali}@cse.shirazu.ac.ir

Abstract—**Malware is a malicious code which is developed to harm a computer or network. The number of malwares is growing so fast and this amount of growth makes the computer security researchers invent new methods to protect computers and networks. There are three main methods used to malware detection: Signature based, Behavioral based and Heuristic ones. Signature based malware detection is the most common method used by commercial antiviruses but it can be used in the cases which are completely known and documented. Behavioral malware detection was introduced to cover deficiencies of signature based method. However, because of some shortcomings, the heuristic methods have been introduced. In this paper, we discuss the state of the art heuristic malware detection methods and briefly overview various features used in these methods such as API Calls, OpCodes, N-Grams etc. and discuss their advantages and disadvantages.**

Keywords-Malware Detection, Computer Security, API Call, N-Gram, OpCode, Control Flow Graph.

I. INTRODUCTION

Malware (short for malicious software), is usually considered as software that aims to disrupt ordinary operations of a computerized system by gathering sensitive information or making unauthorized access to computer systems and mainly harass users [1]. Identifying malwares are critical, due to the fact that nowadays, they are growing increasingly, causing lots of damage to a large surface area.

Malwares can be divided into several categories, such as viruses, worms, Trojans, spywares and adware's, Rootkits, etc. [2]. Virus is a malicious code that can copy itself into other programs. In fact it cannot do anything alone and must be executed by the carrier program which has been infected. Viruses are usually run with user involvement.

Worm is an independent program and do not require any other program for execution. It usually spreads through computer networks and uses system weaknesses for its malicious purposes. It does not require any interference for execution and propagation.

A Trojan, which sometime called as a Trojan horse, is a malware that appears like a legitimate and useful program, but in fact, can compromise computer security and cause much damage.

Spyware is a computer program secretly installed on a personal computer and records or controls what users do with the computer without user's permission. Rootkits are programs set through the influence of the components of the operating system to allow hackers to do malicious things. Sometimes they also influence the operating system kernel.

Backdoor is a malware variant which makes the infected computer ready for remote access without user's permission and does this by opening a backdoor in the victim computer. (i.e. it opens a port)

In signature based malware detection method, doubtful files can be analyzed by matching with the list of available signatures, if a match is found in this comparison, the file under the test will be classified as a malicious executable.

Malware developers apply concealment strategy to avoid detection by signature-based methods. Some malware are changed for each transmission. Some of them encrypt themselves or their malicious activity. Therefor detecting new version of malware, due to the difficulty of extracting their signature that requires time and manpower, made researchers focus on new ways to find more effective and efficient methods. In this work, we investigate the way that malwares conceal themselves and focus on introducing the most effective heuristic methods for malware detection.

The rest of the paper is organized as follows: In section II we briefly discuss an overview of malware detection methods, Section III introduces concealment strategies. In section IV, an overview of researches conducted in heuristic malware detection scope are covered. Section V illustrated a comparison table and finally in section VI the summery of the survey is discussed.

II. MALWARE DETECTION METHODS

Malware detection methods are fundamentally categorized in different categories from different points of view. In this paper we assume three categories for malware detection methods illustrated in Figure 1.

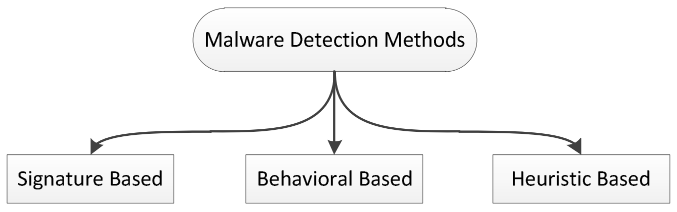


Figure1: Malware Detection Methods

A. Signature-based methods

Nowadays pattern matching is the most common method in malware detection, and signature based detection is the most popular method in this area [3]. Signature is a unique feature for each file, something like a fingerprint of an executable. Signature based methods use the patterns extracted from various malwares to identify them and are more efficient and faster than any other methods. These signatures are often extracted with special sensitivity for being unique, so those detection methods that use this signature have small error rate. Where this small error rate is the main reason that most common commercial antiviruses use this technique [3].

These methods are unable to detect unknown malware variants and also requires high amount of manpower, time, and money to extract unique signatures. These are the main disadvantages of these methods. Also, inability to confront against the malwares that mutate their codes in each infection such as polymorphic and metamorphic one is another disadvantage. To tackle these challenges, research societies propose completely new malware detection family.

B. Behavior-based methods

Behavior based malware detection techniques observe behavior of a program to conclude whether it is malicious or not [4]. Since behavior based techniques observe what an executable file does, they are not susceptible to the shortcomings of signature-based ones. Simply put, a behavior based detector concludes whether a program is malicious by inspecting what it does rather than what it says. In these methods, programs with the same behavior are collected. Thus, a single behavior signature can identify various samples of malware. These types of detection mechanisms help in detecting malware that keep on generating new mutants since they will always use the system resources and services in the similar manner. A behavior-based detector basically consists of the following components [5]:

- **Data Collector:** This component collects dynamic / static information about the executable.
- **Interpreter:** This component converts raw information collected by data collection module into intermediate representations.
- **Matcher:** It is used to compare this representation with the behavior signatures.

One example of a behavior based detection approach is the histogram based malicious code detection technology patented by Symantec [4].

The main advantage of the behavior based malware detection techniques is the ability to detect the type of malwares that signature base techniques are unable to detect such as unknown and polymorphic malware variants. On the other hand, non-availability of promising False Positive Ratio (FPR) and also high amount of scanning time are the main disadvantages of these behavior based malware detection methods [6].

III. Concealment Strategies

In the previous section, we briefly discussed about malware detection strategies used by malware detectors. When malware developers figure out that their malware is going to be detected, try to evade anti malware strategies by applying various concealment techniques. In this section, we introduce some of the most knows concealment strategies.

- **Obfuscation:** In this technique, developers' setout actions that will prevent signature based detection methods to detect their malware. These actions include adding garbage commands, unnecessary jumps etc.
- **Code encryption:** Those malwares use this defensive mechanism encrypt themselves or their malicious activity. Encrypted malware is a complex consists of a decryption algorithm, encryption algorithm, encryption keys and encrypted malicious code. When the malware runs, the key and decryption algorithm have been used to decrypt its malicious part. The malware copies itself and generates a new key, using new generated key and encryption algorithm. A new encrypted version will be produced. This version contains encryption algorithm and the new key. So, even the encryption key and the encrypted code are changing constantly, but they can be detected because of their fixed decoding algorithm.
- **Oligomorphic strategy:** Those malware use this strategy use encryption as a defensive mechanism to protect themselves and are able to change their encryption algorithm for a limited time only. For example, a virus which has a small, finite number of different decryptor loops.
- **Polymorphic strategy:** This type of malware usually encrypts itself by an encryption algorithm. So, in any infection, a different decryption key will be used. Also, a polymorphic malware can use an unlimited number of encryption algorithms in order to avoid detection. In each execution, a part of decryption code will change. Depending on the type of malware, malicious actions or other actions performed by the malware can be placed under the encryption operations. Usually, a transformation engine is embedded in the encrypted malware that at any change generates a random encryption algorithm. Then, the engine and malware are encrypted using the produced algorithm and new decryption key is connected to them.
- **Metamorphic strategy:** Metamorphic malware are the most complex type of malware. These malicious software change themselves so that the new instance has no resemblance to the original one. The malware does not have any coding engine and in each transmission, automatic changes occur in the malware source code.

There are so many different ways to alter the appearance of the malware source code. For example [7]:

- Format alternation removes or adds blanks and comments to the source code. Despite this method is the easiest and the lowest effective among other methods, it sometimes can mislead detection algorithms.

- Variable renaming continuously changes variable identifier name without violating program correctness. Changing the variable names may confuse human but has almost no effect for automated detection techniques.
- Statement reordering can rearrange the sequence of statements in the program if it doesn't generate an error in the program.
- Statement replacement can replace some statements with other statements which have similar functionality if it doesn't make the application logic wrong. This method is more complex than the previous ones.
- Control replacement changes some control structure functions with other control functions which work similarly, for example "for" loop and "while" loop are interchangeable.
- Junk code insertion inserts unimportant code into the program to mislead the diagnosis if it doesn't disrupt the main program logic. In other words, executing junk code does not affect the logic of the source code [1].
- Spaghetti code can distribute or interconnect consecutive statements by unconditional jumps like "goto" command.
- Subroutine inlining and outlining are techniques which replace subroutine's code with subroutine call and vice versa. Code outlining is a technique which replaces subroutine's code with subroutine call. Code inlining is the reverse operation and is a technique employed to avoid subroutine call overhead. These transformations preserve the original code but deal it in different ways.

Regarding to new defensive strategies used by malware authors, in some cases, signature based and behavioral methods are unable to detect protected malware, so a novel method which can efficiently detect this malware is absolutely required.

IV. HEURISTIC METHODS

As we mentioned, signature based and behavior based malware detection methods have some disadvantages. Hence, heuristic malware detection methods are proposed to overcome these disadvantages. Heuristic malware detection methods use data mining and machine learning techniques to learn the behavior of an executable file. For example, as the first attempt, Naïve Bayes and Multi Naïve Bayes [8] were employed by Schultz et al. [8] to classify malware and benign files.

Obviously, these ML techniques must be in classification arena where these methods require some features which represents the input instance in the way that can be used for classification. There are some features investigated for malware detection depicted in Figure 2 which will be described in the followings.

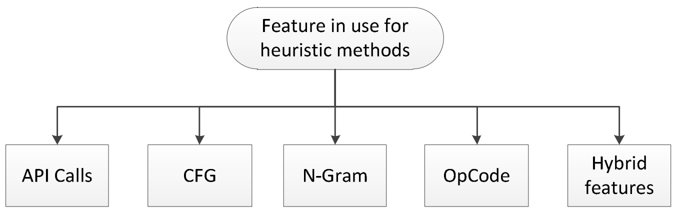


Figure2: Heuristic Methods Features

A. API/System calls

Almost all programs use application programming interface (API) calls to send their requests to the Operating System [9], API call sequences is one of the most attractive way that reflects the behavior of a piece of code like malware.

Hofmeyr et al. were among the first ones who regarded API call sequences as a feature of a malware [10]. They introduced an anomaly detection method based on system call sequences. Normal behavior profiles were made using short sequences of system calls. Hamming distance was used for matching sequences; also a threshold used to determine anomalies. Typically, large Hamming distance value reported as anomalies.

Afterward, an extensive research was made on using API calls such as Bergeron et al. [11], Sekar et al. [12], Sung et al. [13], etc.

In 2007, based on the analysis of Windows API execution sequences called by Portable Executable (PE) files, Ye et al. [14] proposed Intelligent Malware Detection System (IMDS) using Object Oriented Association (OOA) mining based classification. To generate efficient OOA rules for classification, an OOA-Fast-FP Growth algorithm is adapted. Despite its good performance in malware detection, IMDS has two main problems:

1. Handling the large set of generated rules for building the classifier.
2. Finding effective rules to classify new file samples.

To overcome these two problems, Ye et al. [15] used post processing techniques of associative classification. At first they applied Chi-squared testing [16] and insignificant rule pruning, followed by database coverage based on the Chi-square measure rule ranking mechanism and Pessimistic error estimation. They, finally, performed prediction by selecting the best first rule. They incorporate CIDCPF [15] into existing IMDS system and called the new system CIMDS. It was the first attempt on using post processing techniques of associative classification in malware detection.

Jeong and Lee [17] used system call sequences for both malicious and benign executables to build a topological graph which is called code graph. For every binary program this graph is extracted and is compared with the code graph of malicious and benign programs. Based on this comparison, a program will be classified as malware or benign. Unfortunately, this graph is too large. To overcome the disadvantage of large graphs, Lee et al. [18] classified API calls to 128 groups, so the code graph reduced.

Ye et al. [19] proposed an interpretable classifier based on the analysis of API calls by a PE file for detecting malware from large and imbalanced gray list. Their case study were based on 8,000,000 malware, 8,000,000 benign, and 100,000 samples from the gray list collected from an6 Anti-virus lab of King soft corporation. They built effective associative classifier based on several different post processing techniques including rule pruning and rule reordering. Then, to make the classifier less sensitive to the imbalance dataset and improve its performance, they developed the Hierarchical Associative Classifier (HAC).

B. *OpCode*

An OpCode (short for Operational Code) is the subdivision of a machine language instruction that identifies the operation to be executed. More specifically, a program is defined as a series of ordered assembly instructions. An instruction is a pair composed of an operational code and an operand or a list of operands.

The most significant research on OpCodes has been done by Bilar [20]. He showed the ability of single OpCodes to use as a feature in malware detection. To this end, he statistically analyzed the capability of single OpCodes and demonstrated their high reliability for determining the maliciousness of an executable and proved that OpCodes can be used as a powerful representation for executable files.

Santos et al. are the vanguards of malware detection based on OpCodes. They presented various type of malware detection techniques based on OpCode sequences. As an example, in their first work, they presented an approach focused on detecting obfuscated malware variants using the appearance frequency of OpCode sequences in order to build a representation of executable files [21]. To do so, they disassembled the executable files, using the generated assembly files, they have built an OpCode profile that contains a list of OpCodes and after that they compute the relevance of each OpCode based on the frequency of appearance of each of them in both datasets (i.e. malware and benign dataset) using Mutual Information [22]. Finally, they used Weighted Term Frequency (WTF) [23] to make suitable feature vector extracted from executables. They used this feature vector in order to detect obfuscated malware variants and to this end they calculated the Cosine similarity measure between two feature vectors (i.e. New instance feature vector and malware variants feature vector).

Afterward, in the next work, Santos et al. presented a new feature extraction method based on OpCode sequences [23] and trained several machine learning classifiers by embedding the extracted features. As we know, the machine learning based classifiers requires high number of samples for each of the concept classes they try to detect (i.e. malware and benign) and it is quite difficult to obtain this amount of labeled data in real world. So, Santos et al., in their next research, proposed several methods to eliminate this limitation such as Collective classification [24], Single class learning [25], and Semi supervised learning [26].

Runwal et al. [27] proposed a new approach based on OpCodes and used this method for detecting unknown and also metamorphic malware families based on a simple graph similarity measurement. They extracted OpCodes from both file types (i.e. malware and benign), count the number of each pair OpCodes appeared respectively in them and based on the numbers, make a graph of OpCodes and after that can predict the maliciousness of a new executable by calculating the similarity of graph obtained from the executable and both file types and finally the file will be classified as the class which is more similar to.

Shabtai et al. [28] tried to detect unknown malicious codes by applying classification techniques on OpCode patterns. They created a dataset of malicious and benign executables for the Windows operating system. After disassembling the executables, they calculated the normalized term frequency (TF) and TF Inverse Document Frequency (TF-IDF) representations as a feature for each file. Finally, they used several classical classification techniques such as Support Vector Machine (SVM), Logistic Regression (LR), Artificial Neural Networks (ANN) etc. to evaluate the proposed feature selection method.

C. *N-Grams*

N-Grams are all substrings of a larger string with a length of N [29]. For example, the string “VIRUS”, can be segmented into several 3-grams: “VIR”, “IRU”, “RUS” and so on. Over the past decade, several researches have been motivated on the detection of unknown malware based on its binary code content which will be briefly overviewed in this section.

Schultz et al. [8] were the first who introduced the idea of applying ML techniques for detection of diverse malwares based on their own binary codes. Three different feature extraction methods were engaged: features mined from the PE section, expressive plain-text strings that are encoded in executables, and byte sequence features.

Tesauro et al. [30] were the first who try to use N-Grams as a feature for malware detection domain. They used N-Grams to detect Boot Sector Viruses using Artificial Neural Networks (ANN). A Boot Sector Virus is a malware variant which infects DOS Boot Sector or Master Boot Record (MBR). When a system has infected, the MBR is usually ruined and the computer boot order is change. The N-Grams was selected from most frequent sections in malware and benign executables. They used a specific feature reduction algorithm such that each malware must consist of at least four N-Grams from existing N-Grams set.

Tesauro et al. [31], in their next study, used N-Grams to build several classifiers based on ANN and also used a specific voting strategy to achieve final results. In that research a simple threshold value was used to reduce the number of N-Grams.

Abou-Assaleh et al. [29], presented a framework that uses the Common N-Gram method and the K-Nearest-Neighbor (KNN) classifier for malware detection. For both classes (i.e. malicious and benign) a delegate profile was built. A new instance was matched with the profiles of both classes and was assigned to the most similar one.

Kotler and Maloof [32] used byte N-Gram representation to detect unknown malware. Though the vector of N-Gram features was binary, presenting the attendance or nonattendance of a feature in the file. In an extension of their previous study, Kolter and Maloof [33] classified malware into several families based on the functions in their respective payload attempting to approximate their capability to detect malicious codes based on their subject dates.

Cai et al. [34] conducted several experiments in which they evaluated the mixtures of seven feature selection techniques, three classifiers, and byte N-Gram size.

Recently, Moskovitch et al. [35] published the results of a research which used an imbalance data set characterized by byte N-Grams. Moreover, a research of the imbalance problem was illustrated.

D. Control flow graph

Control Flow Graph (CFG) is a graph that represents the control flow of programs and are widely used in the analysis of software and have been studied for many years [36], [37], [38]. CFG is a directed graph, where each node represents a statement of the program and each edge represents control flow between the statements (i.e. what happens after what). Statements may be assignments, copy statements, branches etc. In Figure 4 we can see an example of a generated CFG for Chernobyl malware.

In [39], authors performed a set of normalization operation after disassembling an executable program for reducing effects of mutation techniques and unveiling the flow connections between benign and malicious code. Then they generate corresponding CFG for the program. CFG compared against the CFG of a normalized malware in order to know whether CFG contains a sub graph which is isomorphic to CFG of the normalized one. Thus, the problem of detecting malware is changed to the sub-graph isomorphism problem.

Zhao [40] proposed a detection method based on features of the control flow graph for PE files. At first, he created CFG for each executable file. Then, he used features which extracted from CFG as the train data. These features are information about nodes, edges and subgraphs. After feature selection, some data mining algorithm have been used for classification based on these features such as Decision Tree [41], Bagging [42] and Random Forest [43].

Bonfante et al. [44] used CFG as a signature for malware detection. As we mentioned, CFG is composed of nodes and edges and as we know each assembler consists of four types of instruction: non-conditional jumps (jmp), conditional jumps (jcc), function calls (call) and function returns (ret). They abstract any contiguous sequence of instructions in a node named “inst”, and after that the end of the program comes in a node named “end”. So, they defined six types of node: jmp, jcc, call, ret, inst and end. They build CFG based on these types as illustrated in Figure3. Then, they reduce these nodes in this way: for any node of kind inst or jmp, they removed the node from the graph and linked all its predecessors to its unique successor. After reduction, they used this graph as a signature for each file.

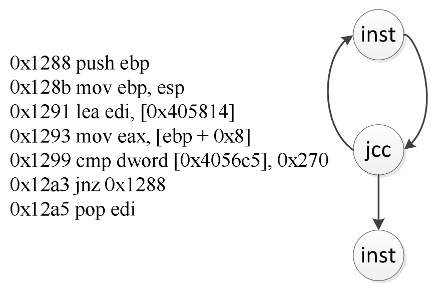

```
0x1288 push ebp
0x128b mov ebp, esp
0x1291 lea edi, [0x405814]
0x1293 mov eax, [ebp + 0x8]
0x1299 cmp dword [0x4056c5], 0x270
0x12a3 jnz 0x1288
0x12a5 pop edi
```

Figure3: CFG Extraction proposed by Bonfante et al. [44].

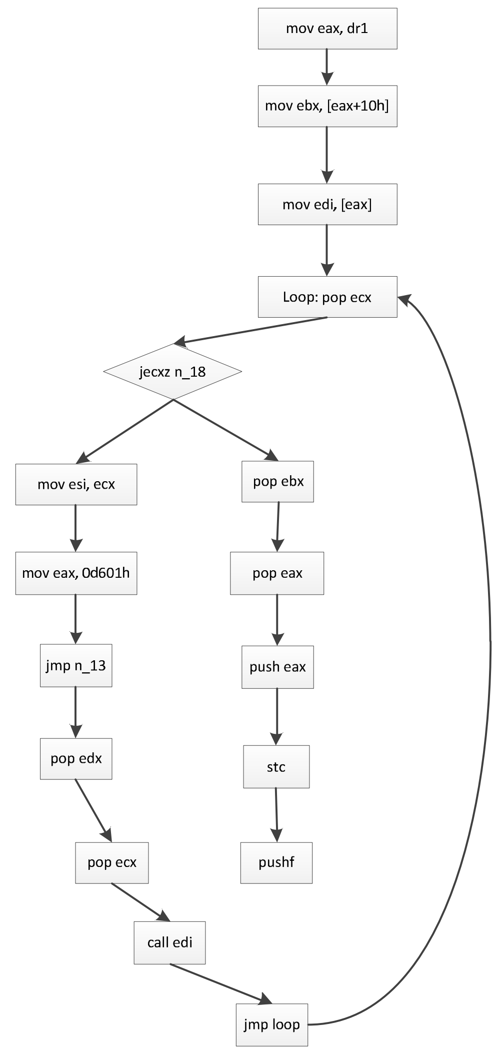


Figure4: An example of Control Flow Graph.

E. Hybrid Features

The performance of machine learning classifiers is influenced by two main factors: features and algorithms. Some researchers set out to improve the precision of machine learning through the features. They combine features to get better accuracy.

As we mentioned, CFG has been successfully applied in detection of simple malware. For detecting complex malware such as polymorphic or metamorphic ones, we must improve CFG based detection methods.

Eskandari et al. [45] used the simple CFG and API calls to detect metamorphic malware. CFG was used to understand semantic of malware. Assuming called API on CFG, they got more semantic aspects of new malware. For decreasing complexity of graph mining algorithms, they converted the sparse graph to the feature vector. The number of generated features is so high. Therefore, for reducing them, they used a feature selection algorithm. Then, they got some data items as training set and generate a set of rules which saved in the database. Decision system utilizes rule database to decide whether each sample of test set is malware or not.

There are two main factors affecting the performance of these methods which are based on ML concepts: features and algorithm used for classification. Lu et al. [46] proposed an accurate system which uses new feature and a novel classifier. For features, they used content based and behavioral based features and combined them. Imported DLLs and API function calls of a PE file was selected as the content based features. However, due to the huge amount of these extracted features, they used the concept of information gain for reducing these features. For behavioral based ones, they execute each file in VMware tool their behavior is represented as a vector. There, they proposed a new ensemble learning method called SVM-AR which combines SVM and Association Rules based on hierarchical taxonomies. Their experimental results showed SVM-AR works better than all popular ensemble learning methods.

F. Some Newly Introduced Features

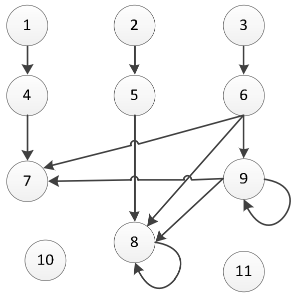
Ye et al. [47] used both file content and file relations as features. By file relations, we mean co-presence or absence of some files in the computer of all cloud users. This relation among file samples provides valuable information about their properties. If an unknown file always co-occurs with many kinds of Trojans in user’s computers, then, most likely, it is a malicious file. File relations among samples can be a novel and practical feature representation for malware detection. They proposed a semi parametric classification model for combining file content and file relation. For file content, they extracted API calls from the content of malicious and benign PE files. Their semi parametric model integrates file content and files relation information and formulates the classification problems using the graph regularization framework. Their case studies on large and real daily malware collection from Comodo Cloud Security Center demonstrate the effectiveness and efficiency of their Valkyrie system and their system has been already incorporated into the Scanning tool of Comodo’s anti-malware software. This is the first work of using both file content and file relations for malware detection.

```
1: dim a, b, c
2: a = 10
3: b = 2
4: for c = 2 to a do
5: b = b + c
6: end for
```

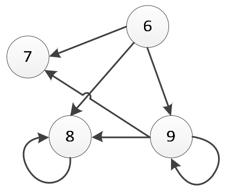
a) VBS Original code

```
 1: dim a
 2: dim b
 3: dim c
 4: set a=10
 5: set b=2
 6: set c=2
 7: if c < a then
 8: set b = b+ c
 9: set c = c + 1
10: goto 7
11: end if
```

b)VBS semantic code



c) dependency graph



d) dependency graph after reduction

Figure 5: An example for semantic code and dependency graph

Kim and Moon [7] proposed a new method to detect polymorphic script viruses. They represented the VBScript malware using a dependency graph. Every script malware code is parsed and transformed to a semantic code. As shown in Figure 5(a)-(b). Based on this semantic code, a directed graph represents relation among the lines of the semantic code that was made. So, each node represents a line of code and each directed edge represents dependency between two lines. In order to decreasing the size of this graph, they omitted nodes and edges that satisfy some conditions. Figure 5(c)-(d) is an example of how a dependency graph is generated. So the system build this dependency graph for a known malware and an unknown file which is not classified yet and compares their dependency graphs to determine whether this new script is a polymorphic variant of that known malware or not. Therefore, the detection is based on finding maximum subgraph isomorphism.

V. METHOD COMPARISON

In this paper we have investigated heuristic malware detection methods. Moreover, a brief overview on advantages, disadvantages and features is given. Indeed, high false positive ratio is the most disadvantage of heuristic malware detection. In this regard, we present a comparison table which is illustrated in Table1.

VI. SUMMERY

In this survey we have presented malware detection methods, and proposed a novel classification scheme for malware detection techniques. We have also a brief overview on inadequacies in the malware detection methods and discuss about how these shortcomings are covered by alternative methods and introduced the strategies used to defeat detection methods called “concealment strategies”. The objective of the survey is to provide a procedure, which could be suitable for further studies and to develop malware detection techniques.

TABLE I: A COMPARISON HEURISTIC MALWARE DETECTION.

Features	Reference	Advantages	Disadvantages
API	[14]	Detects polymorphic and unknown malware. Fewer false positives than other scanners. Outperforms other classification approaches in both detection ratio and accuracy.	Large set of generated rules for building classifier.
	[15]	Outperforms popular antivirus software tools, such as McAfee Virus Scan and Norton Antivirus as well as previous data-mining based detection systems. Reduces the number of Generated rules. Fewer detection time rather than other scanners.	Only provides binary predictions.
	[17]	Detects malware before their execution. Works well under code obfuscation methods.	Large size of graph for comparison.
	[18]	Generates semantic signature. Categorizes API calls to 128 groups to reduce the graph size. Detects metamorphic malware.	
	[19]	Large and imbalanced gray list. Outperforms other classification methods in terms of performance and efficiency. Low time complexity.	
OpCode	[20]	Shows the ability of single OpCodes to use as feature in machine learning malware detection techniques.	
	[21]	Detects obfuscated malware variants.	High number of executable for each of the classes. Imbalance datasets.
	[23]	Extracts proper features for machine learning classifiers.	High number of executable for each of the classes.
	[24], [25], [26]	Does not need high number of executable for each of the classes. Suitable for Imbalance datasets.	It is difficult to evaluate. May be biased.
	[27]	Detects metamorphic malware.	
	[28]	Detects unknown malware. Reduces the false positive rate.	
CFG	[39]	Detects metamorphic malwares.	
	[40]	High detection ratio. Low false positive rate.	Did not compare the efficiency of its algorithm with other techniques.
	[44]	Low false-positives rate.	Did not evaluate false negatives.
CFG API calls	[45]	Detects unknown malwares. Reaches better accuracy than CFG method. Low false positive rate. Less complexity in comparison with dynamic behavior method.	
content-based behavior-based features	[46]	Improves the accuracy of malware detection effectively. Outperforms all popular ensemble learning methods. Reduces the false positive rate.	Takes a long time of training.
file relation file content	[47]	The accuracy of file relation based classifier is similar to the file content based classifier, while combining these two features outperforms each of them. Uses large and real datasets.	
Dependency graph	[7]	Outperforms antiviruses in detecting VBS malwares. Outperforms existing anti-virus software's in Virus Total.	Test the algorithm on small dataset. Not appropriate for PE files. Do not detect all metamorphic techniques. Computational cost of GA.
(PE) section plain-text strings byte sequence	[8]	Detects previously undetectable malicious executable. Detects borderline binaries.	
N-Gram	[34]	Improves the accuracy of malware detection effectively. Low false positive ratio.	Time Complexity.

REFERENCES

[1] J. Aycock. “Computer Viruses and Malware”. Springer, 2006.

[2] P. Szor.” The Art of Computer Virus Research and Defense”. Addison Wesley for Symantec Press, New Jersey, 2005.

[3] P. Gutmann. “The Commercial Malware Industry.”, 2007.

[4] KALPA, “Introduction to Malware”, “http://securityresearch.in/ index.php/projects/malware_lab/introduction-to-malware/8/”, 2011.

[5] G. Jacob, H. Debar, and E. Filiol, “Behavioral detection of malware: from a survey towards an established taxonomy,” Journal in Computer Virology, pp. 251–266, 2008.

[6] A. Ahmed, E . Elhadi, M. A. Maarof and A. H. Osman, “Malware Detection Based on Hybrid Signature Behaviour Application Programming Interface Call Graph Information Assurance and Security Research Group.” Journal, A., Sciences, A., & Publications, S., Faculty of Computer Science and Information Systems, 9(3), 283–288,2012.

[7] K. Kim and B. R. Moon, “Malware detection based on dependency graph using hybrid genetic algorithm.” In Proceedings of the 12th annual conference on Genetic and evolutionary computation, July 07-11, 2010.

[8] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo, ” Data mining methods for detection of new malicious executables.” In IEEE Symposium on Security and Privacy, pages 38-49. IEEE COMPUTER SOCIETY, 2001.

[9] D. Orenstein, “Application Programming Interface (API),” Quick Study: Application Programming Interface (API), 2000.

[10] S. Hofmeyr, S. Forrest, and A. Somayaji, “Intrusion detection using sequences of system calls.” Journal of Computer Security, pp. 151–180, 1998.

[11] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, and N. Tawbi, “Static detection of malicious code in executable programs.” Int. J. of Req. Eng., 2001.

[12] R. Sekar, M. Bendre, P. Bollineni, and D. Dhurjati, “A Fast Automaton-Based Approach for Detecting Anomalous Program Behaviors.” In IEEE Symposium on Security and Privacy,2001.

[13] A. H. Sung, J. Xu, P. Chavez, and S. Mukkamala, “Static Analyzer of Vicious Executables.” In 20th Annual Computer Security Applications Conference, pp. 326–334, 2004.

[14] Y. Ye, D. Wang, T. Li, and D. Ye, “IMDS: Intelligent malware detection system,” in Proc. ACM Int. Conf. Knowl. Discovery Data Mining, pp. 1043–1047, 2007.

[15] Y. Ye, T. Li, Q. Jiang, and Y. Wang, “CIMDS: adapting postprocessing techniques of associative classification for malware detection,” IEEE Trans. Syst., Man, Cybern. C, vol. 40, no. 3, pp. 298-307, 2010.

[16] W. Snedecor and W. Cochran, “Statistical Methods”, 8th ed. Iowa City, IA: Iowa State Univ. Press, 1989.

[17] K. Jeong and H. Lee, “Code graph for malware detection. In Information Networking.” ICOIN. International Conference on, Jan 2008.

[18] J. Lee, K. Jeong, and H. Lee, “Detecting metamorphic malwares using code graphs” In Proceedings of the ACM Symposium on Applied Computing, ser. New York, NY, USA: ACM, pp. 1970-1977, 2010.

[19] Y. Ye, T. Li, K. Huang, Q. Jiang and Y. Chen, “Hierarchical associative classifier (HAC) for malware detection from the large and imbalanced gray list”. Journal of Intelligent Information Systems, 35(1), pp.1-20. 2010.

[20] D. Bilar, “OpCodes as predictor for malware,” International Journal of Electronic Security and Digital Forensics, vol. 1, no. 2, p. 156, 2007.

[21] I. Santos, F. Brezo, J. Nieves, and Y. Penya, “Idea: OpCode-sequence-based malware detection,”, Engineering Secure Software and System , 2010.

[22] C. Peng, H. Long and F. Ding, “Feature selection based on mutual information: cri-teria of max-dependency, max-relevance, and min-redundancy.,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005.

[23] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, “OpCode sequences as representation of executables for data-mining-based unknown malware detection,” Information Sciences, Aug. 2011.

[24] I. Santos, C. Laorden, and P. Bringas, “Collective classification for unknown malware detection,” Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, 2011.

[25] I. Santos, F. Brezo, B. Sanz, C. Laorden, and P. G. Bringas, “Using opCode sequences in single-class learning to detect unknown malware,” IET Information Security, vol. 5, no. 4, p. 220, 2011.

[26] I. Santos, B. Sanz, and C. Laorden, “OpCode-sequence-based semi-supervised unknown malware detection,”, Computational Intelligence in Security for Information Systems , 2011.

[27] N. Runwal, R. M. Low, and M. Stamp, “OpCode graph similarity and metamorphic detection,” Journal in Computer Virology, vol. 8, no. 1–2, pp. 37–52, Apr. 2012.

[28] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, “Detecting unknown malicious code by applying classification techniques on OpCode patterns,” Security Informatics, vol. 1, no. 1, p. 1, 2012.

[29] T. Abou-assaleh, N. Cercone, V. Keˇ, and R. Sweidan, “N-gram-based Detection of New Malicious Code,” no. 1, 2004.

[30] G. B. S. Gerald, J. Tesauro, Jeffrey O. Kephart, “Neural Network for Computer Virus Recognition.” IEEE Expert, 1996.

[31] W. A. and G. Tesauro, “Automatically Generated Win32 Heuristic Virus Detection,” in Virus Bulletin Conference, 2000.

[32] M. M. Kolter JZ, “Learning to detect malicious executables in the wild.” in roc of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006.

[33] J. Z. Kolter and M. A. Maloof, “Learning to Detect and Classify Malicious Executables in the Wild,” vol. 7, pp. 2721–2744, 2006.

[34] T. J. Cai DM, M. Gokhale, “Comparison of feature selection and classification algorithms in identifying malicious executables,” in Computational Statistics and Data Analysis, 2007.

[35] E. Y. Moskovitch, D. Stopel, C. Feher, N. Nissim and N. Japkowicz, “Unknown malcode detection and the imbalance problem,” journal in Computer Virology, 2009.

[36] P. Jalote, "An Integrated Approach to Software Engineering", Springer, New York, NY, 2005.

[37] T. McCabe, "A complexity measure", IEEE Transactions on Software Engineering SE-2(4): 308–320, 1976.

[38] L. Tan, "TheWorst Case Execution Time Tool Challenge", The External Test, Technical report, 2006.

[39] D. Bruschi, L. Martignoni and M. Monga “Detecting self-mutating malware using control-flow graph matching,” In: Büschkes, R. and Laskov, P. (eds) Detection of Intrusions and Malware & Vulnerability Assessment, volume 4064 of LNCS, pp 129–143. Springer, Berlin. 2006.

[40] Z. Zhao, “A virus detection scheme based on features of Control Flow Graph.” 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), pages 943-947, 2011.

[41] T. M. Mitchell, “Machine learning and data mining,” *Commun. ACM,* vol. 42, no. 11, 1999.

[42] L. Breiman. “Bagging Predictors.” Machine Learning, 24(2):123–140, 1996.

[43] L. Breiman. “Random Forests.” Machine Learning, 45(1):5–32, 2001.

[44] G. Bonfante, M. Kaczmarek, J.Y. Marion. “Control Flow Graphs as Malware Signatures.” WTCV, May, 2007.

[45] M. Eskandari and S. Hashemi “Metamorphic malware detection using control flow graph mining”. International Journal of Compute Science Network Security,pp 1-6 ,2011.

[46] Y. Lu, S. Din, C. Zheng and B.Gao “Using multi-feature and classifier ensembles to improve malware detection”. Journal of CCIT 39(2), 57–72. 2010.

[47] Y. Ye, T. Li, S. Zhu, W. Zhuang, E.Tas, U. Gupta and M. Abdulhayoglu, “Combinig File Content and File Relations for Cloud Based Malware Detection.” Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, 2011.