A Survey on Mining Program-Graph Features for Malware Analysis

Md. Saiful Islam[1], Md. Rafiqul Islam[2], A.S.M. Kayes[1], Chengfei Liu[1], and Irfan Altas[2]

[1] Swinburne University of Technology, Australia
[2] Charles Sturt University, Australia
{mdsaifulislam@swin,mislam@csu,akayes@swin,cliu@swin,ialtas@csu}.edu.au

Abstract. Malware, which is a malevolent software, mostly programmed by attackers for either disrupting the normal computer operation or gaining access to private computer systems. A malware detector determines the malicious intent of a program and thereafter, stops executing the program if the program is malicious. While a substantial number of various malware detection techniques based on static and dynamic analysis has been studied for decades, malware detection based on mining program graph features has attracted recent attention. It is commonly believed that graph based representation of a program is a natural way to understand its semantics and thereby, unveil its execution intent. This paper presents a state of the art survey on mining program-graph features for malware detection. We have also outlined the challenges of malware detection based on mining program graph features for its successful deployment, and opportunities that can be explored in the future.

Key words: Program Graph, Graph Features, Malware Detection

1 Introduction

Malwares are one of the most severe problems witnessed by the modern computer society everyday. Malware is a malevolent software that either tries to disrupt the normal computer operation or gather sensitive information from private computer systems by spying on users' behavior and compromising their privacy. The malware writers also apply various code obfuscation techniques on previous malwares, changing their internal structures while keeping their original functionalities unchanged known as *polymorphic malwares*, to evade detection. The obfuscation techniques also facilitate widespread proliferation of various instances of the same malware without getting detected. David Perry from Trend Micro reported that some antivirus (AV) vendors are seeing 5,000 distinct malware samples per day [33]. A malware detector determines the malicious intent of a program and thereafter, stops its execution if it is malicious. In malware detection based on mining graph features, a program graph is constructed by considering System and/or API calls, sub-programs and targets of jump instructions as nodes and their calling relationships as connections or links. A program

graph may come in various formats and names, e.g., call graphs, control-flow graphs, code graphs etc. Once constructed, a program graph can be mined to extract important graph-based-features which can be intelligently learned into a classifier to detect malicious intent of the unknown program.

While a substantial number of malware detection techniques based on static and dynamic analysis has been studied for decades ([13], [23] for survey), malware detection based on mining program graph features is not established yet. However, it is commonly believed that graphs can be used to represent complex program behavior efficiently [37], which may not be possible in traditional static and dynamic behavior analysis [12], [19]. For example, code obfuscations are easily detectable through mining program graph features. Therefore, graph based representation of program behavior and thereby mining important graph features for malware detection has attracted recent attention [27], [42],[12], [7], [19], [29], [8], [24], [11], [1], [16], [35], [10]. Though, a number of research efforts has been made for malware detection based on mining program graph features, this is still in its premature stage. In this survey paper, we report only representative and important state of the art research works that develop malware detection techniques based on mining program graph features. We also provide necessary background on malware detection based on program graph analysis and outline important research directions that can be explored in the future.

The main contributions of this survey paper are given as follows:

- We provide the background on the program graph, its construction and mining program graph features for malware detection.
- We present a survey on representative works that develop malware detection techniques based on mining program graph features.
- We provide a comparative study of the existing works under the variants of program graph and a summary of them.
- We outline the future challenges that need to be addressed for successful deployment of malware detectors based on mining program graph features.

The rest of the paper is organized as follows: Section 2 presents related surveys; Section 3 provides preliminaries on program graphs and mining program graph features; Section 4 describes the modules of a generalized malware detection system based on mining program graph features; Section 5 presents the surveyed works; Section 6 presents the limitation of the existing works and the future challenges of malware detection based on mining program graphs; and finally, Section 7 concludes the paper.

2 Related Surveys

Survey on Malware Analysis Techniques: In [38], Siddiqui et al. present a survey on malware detection based on data mining techniques. The surveyed works learn the detection model by mining the file features extracted from the binary programs. The authors categorize the surveyed works based upon the analysis type (static and dynamic) and detection type (misuse and anomaly).

In [36], Shabtai et al. present a taxonomy of malware detection methods that rely on machine learning classifiers (which utilize static features extracted from executables) to detect malwares. The authors also address various aspects of the detection challenge such as file representation and feature selection methods, classification algorithms, weighting ensembles, the imbalance problem, active learning and chronological evaluation. In [18] Felt et al. survey the current state of mobile malware in the wild. The authors collect malware samples from iOS, Android, and Symbian mobile platforms to analyze the incentives and possible defenses of them. They classify collected malwares by analyzing their behavior and describe the incentives that actually promote each type of malicious behavior. They also present defenses that disincentivize some of the behaviors of these malwares. In [14], Egele et al. provide an overview of techniques based on dynamic analysis that are used to analyze potentially malicious samples.

Survey on Code Obfuscation Techniques: In [3], Balakrishnan and Schulze analyze the different code obfuscation techniques in connection with the protection of intellectual property and the hiding of malicious code. In [31], Majumder et al. provide an overview of obfuscation and highlight that the dispatcher model and opaque predicates transforms have provable security properties. In [47], You and Yim present a survey on malware obfuscation techniques by reviewing the encrypted, oligomorphic, polymorphic and metamorphic malwares.

Survey on Botnet Detection Techniques: A botnet is a collection of compromised computers, which are remotely controlled by hackers to perform various network attacks, such as denial-of-services and information phishing. In [17], Feily et al. present a literature survey on botnet and four classes of botnet detection techniques: (a) signature-based (b) anomaly-based (c) DNS-based and (d) mining-based. In [2], Bailey et al. provide a brief survey on existing botnet research, the development and trends of botnets as well as different types of networks that approach the botnet problem with differing goals and visibility. In [49], Zhang et al. survey the latest botnet attacks and defenses by introducing the principles of fast fluxing and domain fluxing, and explain how these techniques were employed by the botnet owners. In [39], Silva et al. presents a comprehensive survey on the botnet problem by briefly summarizing the previous works from the literature. Then, the authors supplement these works by covering an extensive range of discussion of recent works and providing solution proposals.

Survey on Intrusion Detection Techniques: In [20], Garcia-Teodoro et al. present a survey of the most well-known anomaly-based intrusion detection techniques. The authors also present the available platforms, systems under development and research projects in the area. Finally, the authors outline the main challenges that need to be addressed for the wide scale deployment of anomaly-based intrusion detectors. In [41], Tavallaee et al. review the experimental practice in the area of anomaly-based intrusion detection by surveying 276 studies published during the period of 2000-2008. Finally, the authors summarize their observations, e.g., why anomaly-based intrusion-detection methods

are not adopted by the industry and identify the common pitfalls among the surveyed works.

To the best of our knowledge, there exists no other paper that provides a complete survey on program graphs and malware analysis based on mining graph features. The purpose of this paper is to provide a tutorial on program graphs and mining graph features, as well as survey the existing works that extract program graph features for malware analysis. We also summarize the limitation of existing works and give challenges to explore further in this direction.

3 Preliminaries

In this section, we present the generalized definition of program graph, properties and common features mined on such graphs.

Definition 1. *(**Program Graph**) A program graph, denoted by g, is represented as $g = (V, E, l_v, l_e)$, where V is the set of nodes and E is the set of links. A directed link (u, v) will be in E if there is a directed relationship from u to v, where both u and v is in V. The l_v denotes a labeling function that returns the label of the nodes in V, i.e, $l_v : V \rightarrow \Sigma_v$. The l_e denotes another labeling function that returns the label of the links in E, i.e, $l_e : E \rightarrow \Sigma_e$. A path p is a sequence of nodes $v_1, v_2, ..., v_k$ such that there is a link between the successive pair of nodes in this sequence. If the start and end nodes are the same, we call it a cycle c.*

The nodes in V may consist of system or API calls, targets of jump instructions, procedure calls and jump targets of a program, even a code fragment of a program including program instructions itself [25], [7], [29], [16]. The links in E may represent the calling relationships (i.e, who calls whom) and/or information flows between nodes. The nodes and links are summarized in Table 1.

Type	**Value(s)**
Nodes	System&API Calls, Procedure Calls, Jump Instructions, Targets of Jump, Code Fragments, Program Instructions
Links	Calling Relationships, Information Flows

Table 1. Nodes and Links of Program Graphs

The generalized definition of program graph given in Definition 1 can be made specific for call graphs, control-flow graphs (CFGs), code graphs etc. as shown in Figure 1. The nodes and links in different variant of program graphs are shown in Table 2. An example of a program call graph (who-calls-whom), which is directed, is shown in Figure 2.

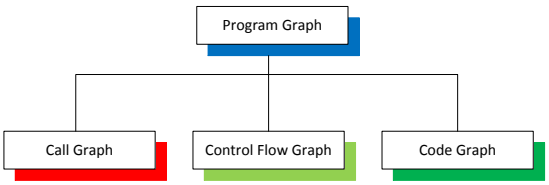

Fig. 1. Variants of program graph

Table 2. Nodes and links in different program graphs

Program Graph	**Nodes**	**Links**	Σ_v	Σ_e
Call Graph	Sub-functions, API/System Calls	$u \rightarrow v$ (u calls v), where u, v are nodes	Sub-function and API or System Call names	$l_v(u) \rightarrow l_v(v)$, where (u, v) is a link
Control-Flow Graph	Sub-functions, API/System Calls, Targets of jump-instructions	$u \rightarrow v$ (u calls v or u jumps to v), where u, v are nodes	Sub-function and API or System Call names, Jumps	$l_v(u) \rightarrow l_v(v)$, where (u, v) is a link
Code Graph	Opcodes or Instructions	$u \rightarrow v$ (v follows u), where u, v are nodes	Opcode or Instruction names	$l_v(u) \rightarrow l_v(v)$, where (u, v) is a link

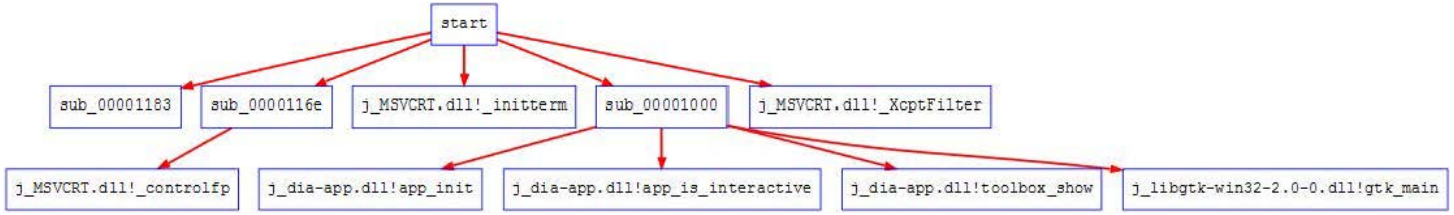

Fig. 2. A sample call (who-calls-whom) graph

The root node "start" in the call graph is the node from where execution begins and is the same for every program call graph. The success of a malware detector based on mining program graph features depends on the accurate construction of the corresponding program graph and the information encoded in it [28], [25]. In [28], Kruegel et al. propose an approach for identifying the program control-flow-graph and program's instruction information based on its initial control flow graph and statistical methods. Generally, the construction of program graph is accomplished as a preprocessing step by the detectors [25].

Definition 2. *(**Graph Isomorphism**) Given two program graphs $g_1 = (V_1, E_1, l_v, l_e)$ and $g_2 = (V_2, E_2, l_v, l_e)$, g_1 is said to be subgraph isomorphic to g_2, if there is an injective function $\mu : V_1 \rightarrow V_2$ such that (1) $\forall u, v \in V_1$ and $u \neq v$, we have $\mu(u) \neq \mu(v)$. (2) $\forall v \in V_1$, we have $\mu(v) \in V_2$ and $l_v(v) = l_v(\mu(v))$. (3) $\forall (u, v) \in E_1$, $(\mu(u), \mu(v)) \in E_2$ and $l_e((u, v)) = l_e((\mu(u), \mu(v)))$. We say that g_1 is a subgraph of g_2 and g_2 is a supergraph of g_1. Given two program graphs g_1 and g_2, graph g_{12} is said to be a common subgraph of g_1 and g_2, if g_{12} is subgraph isomorphic to both g_1 and g_2, respectively.*

Proving subgraph isomorphism is NP-complete [21], which urges for approximate matching between program graphs in large datasets. To facilitate this approximate matching for nodes and links between pair of program graphs, the node label function l_v and link label function l_e can be enhanced to return additional information about nodes and links beyond their labels. Approximate/inexact graph matching also urges an efficient indexing mechanism in the databases for storing program graphs. The advanced graph indexing techniques proposed elsewhere [51], [44] can be borrowed to serve this purpose.

Definition 3. *(**Graph Database**) The database D is a collection of program graphs, $\{g_1, g_2, ..., g_n\}$, where each g_i can have either labels "malware" (m) or "benign" (b), i.e., $l_g : D \rightarrow \Sigma_g$, where $\Sigma_g = \{m, b\}$ and l_g is a labeling function.*

Definition 4. *(**Graph Pattern**) A graph feature, f_i, is a pattern mined from program graphs in D that is highly discriminative for characterizing samples in either class. A pattern space, F, is a collection of highly discriminative patterns.*

In general, graph features are extracted by applying special techniques on nodes, links, paths and cycles of the program graph, and thereby, to generate signatures that can represent the corresponding program efficiently. The performance of the detector system largely depends on the intelligent selection of features and/or construction of signatures [19], [16].

Definition 5. *(**Malware Detection Based on Mining Program Graph Features**) Given the database of program graphs (D) and the pattern space (F), a malware detector learns a classifier or model, C, that can correctly return the label of an unknown program graph.*

By learning a classifier or model in the above definition we mean gathering enough information (i.e., graph features) from existing graphs in D so that C can classify an unknown program as either *malware* or *benign*. This can be modeled as both *lazy* and *eager* learning approach or a combination of both for developing a multi-step/multi-evidence based malware detectors [48]. For example, we can adopt *lazy* learning approach for k-nearest neighbor (kNN) classification where an unknown program is labeled based on the labels of its k nearest neighbors [50]. Though, we can apply eager learning approach for learning the appropriate value of k from D. In classifiers like SVM, we need to adopt *eager* learning approach to learn an appropriate weight function W from D for classifying an unknown program graph [46]. The adoption of a specific learning approach depends on the learned classifier used by the detector system.

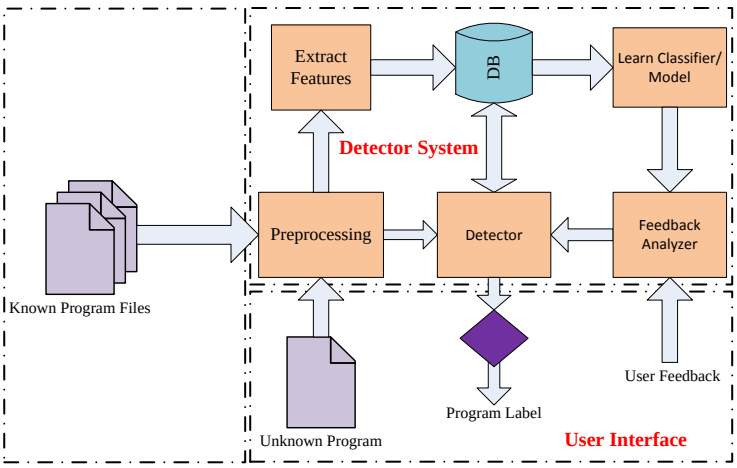

Fig. 3. Conceptual block diagram of a generalized malware detection system based on mining program graph features

4 A Generalized Malware Detection System based on Program Graph Analysis

We depict a generalized conceptual framework of a malware detection system based on mining program graph features in Figure 3. The framework shows important modules and the information flow between them by dividing the whole system into two main parts: (a) detector system and (b) user interface. The detector system classify or label an unknown program based on the learned classifier/model in general. The purpose of user interface is to include user feedback in the detection system to make it more usable and transparent to end user [40].

The functions of individual modules are described below in detail:

- **Preprocessing:** This module preprocess and construct program graphs from program files. To do so, this module first dissemble the input binary (e.g., program executables) and then, gather information to construct the corresponding program graph (e.g., call graph, control-flow graph, code graph etc.).
- **Extract Features:** This module extract features by mining program graphs and store them into the database (DB). The database may be equipped with specialized data indexing techniques to facilitate efficient access.
- **Learn Classifier/Model:** This module learn classifier/model from features stored in DB. This module may rerun if there is a need to tune performance of the detector system.
- **Feedback Analyzer:** This module analyze user feedback to facilitate feedback driven detection and performance tuning of the system.
- **Detector:** This module classify and label an unknown program based on learned classifier/model and analyzed user feedback.

In this paper, we present a brief survey of the state-of-the-art important works under the variants of program graph and the generalized program graph-based malware detection system shown in Figure 3.

5 Surveyed Works

This section presents the surveyed works under the variants of program graph and the generalized graph-based malware detection system shown in Figure 3.

5.1 Control-Flow Graph Based Systems

- In [43] Wagner and Dean propose a technique in which a control flow graph for a program is constructed from its system call trace. Then at run time, this graph is compared with the known system call sequences to check for any violation. The authors apply both static analysis and dynamic monitoring to combat malwares and claim that this combination yields better results.
- In [5], Bruschi et al. propose a strategy that can detect the metamorphic malicious code inside a program P. To do so, they compare the control flow graphs of P against the set of known malwares' control flow graphs. Firstly, they unveil the flow connections between the benign and the malicious code within P after disassembling and performing a set of normalization operations on P. Then, they build the corresponding *labeled inter-procedural control flow graph* for P. This control flow graph of P is then compared against the control flow graphs of known malwares to see whether P contains a subgraph isomorphic to the control flow graphs of the known malwares. The authors also provide an encouraging experimental results to support their approach.
- In [27], Kruegel et al. propose an approach for detecting structural similarities between variations of a polymorphic worm based on control flow information. A fingerprinting technique is developed based on a coloring scheme of the control flow graph, which characterizes the high-level structure of a worm executable. The authors claim that their proposed system is more robust to polymorphic modifications of a malicious executable and is capable of detecting some previously unknown, polymorphic worms.
- Cesare and Xiang [7] propose a system for detecting polymorphic malwares using control-flow graphs. They apply an existing approximated flow graph matching algorithm [6] to estimate graph isomorphisms at the *procedure level*. The similarity between programs is then quantified by identifying the underlying isomorphic flow graphs. Firstly, the approach applies depth-first ordering technique to order the nodes in the control flow graph. Then, a signature is constructed as a list of graph links for the ordered nodes, where the node ordering is used as node labels. This signature is represented as a string. To classify an unknown program, Dice coefficient is computed as the similarity score between the set of the flow graph strings of the unknown program and each set of flow graphs associated with malware stored in the database.

- In [8] Cesare and Xiang propose a similarity search technique for malwares using novel distance metrics. A malware signature is described by a set of control flow graphs contained in the malware. Then, a feature vector is constructed by decomposing these control flow graphs into either fixed size k-subgraphs, or q-gram strings of the decompiled high-level source. A distance metric between two sets of control-flow graphs is then computed based on the minimum matching distance between the corresponding feature vectors of the sets. The minimum matching distance utilizes the string edit distance and the minimum sum weight matching between two sets of graphs. The authors claim that the above technique runs in real time and detects more malware variants in comparison with other existing malware variant detection techniques.
- In [15] and [16], Eskandari and Hashemi propose a control-flow graph based malware detection method by converting the sparse matrix of the control-flow graph into a vector where they save only the situations of nonzero items. Then, a feature vector for each program graph is constructed by taking the API number as data-item and edge number as feature. Among different classifiers, the authors observe that random forest attains best result.
- In [10] Cesare et al. propose "Malwise" for malware classification. A fast application-level emulator is used to reverse engineering the code packing transformation. To classify a malware, two flow-graph based matching techniques are proposed. The exact flow-graph matching algorithm adopts string-based signatures. Firstly, they use depth first ordering technique to order the nodes in the control flow graph. A signature subsequently consists of a list of graph links for the ordered nodes, where the node ordering is used as node labels. Then, the similarity between two flow graphs with signatures x and y is calculated as 1 iff $x = y$, otherwise 0. The approximate flow graph matching algorithm uses string edit distance to quantify the similarity between two flow graphs. To generate string-based signatures, they apply the decompilation technique of structuring. The intuition is that malware variants share similar high-level structured control flow. The authors claim that the probability of detecting the new malware as the variant of existing malware is 88%.

5.2 Code Graph Based Systems

- In [25], Jeong and Lee proposed a code graph based malware detection system by analyzing the instructions related to the system-call sequence in binary executables and then demonstrating their outcomes in the form of a topological graph. These topological graphs are used to preview the effects of programs on a system. Application programs are then tested with the code graph system to extract their distinctive characteristics. These distinctive characteristics are then used to separate malwares (worms and botnets) from normal programs. The authors claim to detect 67% of unknown malwares from normal programs.
- In [1], Anderson et al. present a code graph-based malware detection technique by dynamically collecting instruction traces of the executables. The instructions represent nodes in the graph. To transform the graphs into Markov

chains, the links are labeled with transition probabilities, where the transition probabilities are estimated from the data contained in the trace. A similarity matrix between the graphs is constructed by using a combination of graph kernels. To perform the classification, the similarity matrix is then sent to a support vector machine. The above technique significantly outperforms the signature-based and many other machine learning-based detection methods.
– Runwal et al. [35] present a method for computing the similarity of executable files, based on opcode graphs. This opcode graph based technique is similar but simpler than the one based on instruction trace graphs proposed in [1]. Instead of using graph kernels to generate scores and SVMs for classifications, the authors directly compare the opcode graphs and compute the similarity scores. The authors then propose to apply these similarity scores to detect metamorphic malwares and claim that their approach can outperform a previously developed technique in [45] based on hidden Markov models.

5.3 Call Graph Based Systems

– In [4], Bergeron et al. propose a malware analysis technique based on program graph, which consists of three major steps. Firstly, the binary code of the program is transformed into an internal intermediate form. Secondly, the intermediate form is converted to various relevant graphs, e.g., control-flow graph, data-flow graph, call graph and critical-API graph via flow-based analysis. Finally, these graphs are checked and verified against the security policy.
– Lee et al. [29] propose to reduce the call graphs of malwares into code graphs for extracting semantic signatures. To do so, they consider only API calls in the call graphs. They produce 128 groups (32 objects 4 behaviors) to reduce the sizes of the call graphs. During this reduction step, links represented call relationships are maintained. Finally, a code graph is represented by a 128×128 adjacent matrix and saved. To estimate the similarity between two code graphs, they divide the number of links of the union graph with the number of links of the intersection graph. A suspicious program is identified by computing its code graph similarity score with the code graphs of the known malwares. The authors claim that their proposed mechanism achieves 91% detection ratio of real-world malwares and detects 300 metamorphic malwares that can evade anti-virus (AV) scanners.
– In [22] Han et al. propose a metamorphic malware classification method using the sequential characteristics of API calls used. The authors also present experimental results using the proposed method with some malware samples.

5.4 Other Graph Based Systems

– Fredrikson et al. [19] present an automatic technique for extracting optimally discriminative specifications, which can be used by a behavior-based malware detector based on graph mining and concept analysis. To do so, they first divide the positive (benign) set of programs into disjoint subsets of behaviorally

similar programs. Then, a dependence-graph is constructed for each malware and benign application to represent its behavior. Then, significant behaviors specific to each positive subset are mined. A significant behavior is a sequence of operations that distinguishes the programs in a positive subset from all of the programs in the negative (malware) subset. The author use *structural leap mining* to identify multiple distinct graphs that are present in the dependence graphs of the positive subset and absent from the dependence graphs of the negative set. The significant behaviors mined from each positive subset are combined via *concept analysis* to obtain a discriminative specification for the whole positive set. A specification is said to be discriminative if it matches malicious programs but does not match benign programs and therefore, can be used in the detection of unknown malware. The authors claim to achieve an 86% detection rate on new, unknown malware, with 0 false positives.

- The authors in [24] present a system called JACKSTRAWS, which automatically extracts and generalizes graph templates to capture the core of different kinds of command and control (C&C) activities. Then, these C&C templates are matched against the behavioral graphs produced by other bots. Firstly, the authors record the activities (e.g., system calls) on the host system that are related to data that is sent over and received via each network connection. These activities are then used to construct the behavioral graphs. One graph is constructed for each connection. Then, all behavioral graphs that are constructed during the execution of a malware sample are checked against the templates of different types of C&C communication. When a sufficiently close match is found, the corresponding connection is reported as C&C channel. The authors claim that JACKSTRAWS can accurately detect C&C connections, even for bot families that were not previously used to generate the templates.
- The authors in [11] present Polonium, which is a novel semantic technology for detecting malicious programs via large-scale graph inference. Polonium applies scalable belief propagation algorithm to compute the reputation of program files and program files with low reputations are identified as malwares. They generate an undirected and unweighted bipartite machine-file graph from the raw data. A link exists between a file and a machine that has the file. The links are unweighted and there exists at most one link between a file and a machine. The algorithm predicts the label of a node from some prior knowledge about the node and from its neighbor nodes. The idea is that good files are supposed to appear on many machines and bad files appear only on few machines.

6 Future Challenges

This section presents the future challenges of mining program graphs under the generalized malware detection framework depicted in Figure 3.

6.1 Efficient Construction of (Lossless) Program Graph

A malware can damage the host as soon as it starts its execution. Therefore, the most effective means of protecting the host system is to detect and block

the malware before it starts executing. Graph based representation of program is a natural way to understand its semantics [19] and also facilitate unveiling its execution intent [43]. In connection with this, a program graph must satisfy the following properties: (a) it should characterize the program accurately and include all important information needed by the feature extraction module of the detectors to reduce the false positives/negatives; (b) it should be easily comprehensible by a human being to facilitate user feedback in the system; (c) it should be easy to identify the similarity/dissimilarity of multiple program graphs to increase the accuracy and also for detecting polymorphic malwares; and (d) it should be easy to distinguish the program graphs of different groups to separate malwares from benign programs. Most of the existing works propose detection techniques that apply various noise reduction/node summarization techniques (e.g., [29]) and are therefore, lossy. It should be noted that properties (c) and (d) are quite hard to ensure in the program graph (*one diminishes the other*) and therefore, a very challenging problem. The study made by Cesare and Xiang in [9] may help towards addressing the above problem.

6.2 Exact Matching vs. Approximate Matching

In exact graph matching, we test whether a program graph is a subgraph or supergraph of another program graph and is important for detecting polymorphic malwares [7]. However, testing subgraph or supergraph isomorphism is an NP-hard problem [21]. This problem becomes more hard as polymorphic malwares add noise (e.g., obfuscated codes) into the original version of the malware program, which urges an efficient noise reduction techniques before feeding the program graph to the matching algorithm and also, approximate matching techniques between program graphs in large datasets. To facilitate approximate matching for nodes and links between pair of program graphs, the node labeling function l_v and link labeling function l_e in the program graph g can be enhanced to return additional information about nodes and links beyond their labels. For example, the sub-function nodes in the program graph can itself represent another sub-program graph which can be used to match sub-function node of another program graph that has a different node label. Approximate graph matching also urges an efficient indexing mechanism in the databases for storing program graphs. The approximate matching techniques proposed in the existing works are either insufficient or inapplicable in large scale datasets. The advanced graph indexing techniques proposed elsewhere [51], [44] can be borrowed to serve this. We can also transform the pair-wise similarity problem into a similarity search problem over the database.

6.3 Behavioral Patterns and Malware Signatures

The success of malware detection based on mining program graphs depends on discovering discriminative and important behavioral program graph patterns/features that can separate malwares from benign programs [19]. The system-call calling sequences encoded in the program graph paths and information encoded in it can be exploited to serve this. Frequent paths can be

treated as behavioral patterns. However, polymorphic codes may transform the frequent path to be infrequent. To solve this, a path can be represented by a string and thereafter, approximate string matching [32] along with noise reduction [29] techniques can be applied to find frequent paths from the program graphs. Also, frequent-subgraph idea can be implemented to discover frequent patterns and thereby, to identify metamorphic code in the program. The authors in [26] propose to relax the rigid structure constraint of frequent subgraphs by introducing connectivity to frequent itemsets, which can eases the detection of metamorphic code in the malware program. The malware detectors must also conform properties such as soundness and completeness [34] of them. The challenge is to incorporate these techniques in an integrated manner. None of the existing techniques works in this direction.

6.4 Non-executable Code

There are worms that do not rely on executable codes, rather these worms are written in non-compiled scripting languages. These kind of worms urge specialized techniques to detect their behavioral patterns through program graph. We believe that the generalized program graph construction techniques [37], [19] and features of binary programs [9] can be customized to serve this purpose. An obvious challenge is to analyze malware program in heterogeneous environments e.g., matching a polymorphic malware in a platform that is different from the known malware's platform (i.e., inter-platform comparability).

6.5 Other Graphs

There are works based on graphs, other than the control-flow or code-graphs, utilizes information not only encoded in the program itself, but also its host [11] and its command and control activities [24]. A hybrid approach can be implemented to have the positive aspects of both of these techniques.

6.6 User Feedback

To the best of our knowledge, user feedback is not studied in graph based malware detection techniques. Li et al. [30] develop a malware (virus) detection technique with *real-valued negative selection* (RVNS) algorithm. The authors propose to utilize the arguments of process calls to train the detector and integrate user feedback for tuning the threshold between normal files and viruses. User feedback can be used not only for improving the system performance as described above but can also be tied to improve the usability and transparency of the system. Stumpf et al. [40] demonstrate that it is possible to improve the accuracy of the machine learning system as well as gain the trust of the users by gathering various forms of user feedback, e.g., collecting user feedback on *why* the prediction was wrong after explaining *how* the reasoning made by the system. We propose to utilize such forms of user feedback from graph construction (e.g., nodes/links selection) to feature and system's parameters selection. The challenge is to develop a model that requires minimal user involvement and is also capable of integrating it to tune the model parameters successfully.

7 Conclusion

This paper presents a brief survey on malware detection techniques based on mining program graph features. We have outlined the variants of program graph, their properties and presented the surveyed works under them. We have also presented the challenges that have not been addressed yet as future research direction. To the best of our knowledge there are no other surveys on malware detection based on program graph analysis. We believe that the tutorial and future research challenges presented in this paper may serve as the collective knowledge base among the malware research community.

8 Acknowledgement

M.S. Islam and C. Liu are supported by the Australian Research Council (ARC) discovery project no. DP140103499.

References

1. Anderson, B., Quist, D., Neil, J., Storlie, C., Lane, T.: Graph-based malware detection using dynamic analysis. Journal in Computer Virology 7(4), 247–258 (2011)
2. Bailey, M., Cooke, E., Jahanian, F., Xu, Y., Karir, M.: A survey of botnet technology and defenses. In: Cybersecurity Applications & Technology Conference for Homeland Security. pp. 299–304. IEEE Computer Society, Washington, DC, USA (2009)
3. Balakrishnan, A., Schulze, C.: Code obfuscation literature survey (2005)
4. Bergeron, J., Debbabi, M., Desharnais, J., Erhioui, M.M., Lavoie, Y., Tawbi, N.: Static detection of malicious code in executable programs. Int. J. of Req. Eng (2001)
5. Bruschi, D., Martignoni, L., Monga, M.: Detecting self-mutating malware using control-flow graph matching. In: Bschkes, R., Laskov, P. (eds.) Detection of Intrusions and Malware and Vulnerability Assessment, Lecture Notes in Computer Science, vol. 4064, pp. 129–143. Springer Berlin Heidelberg (2006)
6. Carrera, E., Erdélyi, G.: Digital genome mapping–advanced binary malware analysis. In: Virus Bulletin Conference (2004)
7. Cesare, S., Xiang, Y.: A fast flowgraph based classification system for packed and polymorphic malware on the endhost. In: AINA. pp. 721–728 (2010)
8. Cesare, S., Xiang, Y.: Malware variant detection using similarity search over sets of control flow graphs. In: TrustCom. pp. 181–189 (2011)
9. Cesare, S., Xiang, Y.: Static analysis of binaries. In: Software Similarity and Classification, pp. 41–49. SpringerBriefs in Computer Science, Springer London (2012)
10. Cesare, S., Xiang, Y., Zhou, W.: Malwise - an effective and efficient classification system for packed and polymorphic malware. IEEE Trans. Computers 62(6), 1193–1206 (2013)
11. Chau, D.H., Nachenberg, C., Wilhelm, J., Wright, A., Faloutsos, C.: Large scale graph mining and inference for malware detection. In: SDM. pp. 131–142 (2011)
12. Chen, C., Lin, C.X., Fredrikson, M., Christodorescu, M., Yan, X., Han, J.: Mining graph patterns efficiently via randomized summaries. PVLDB 2(1), 742–753 (2009)

13. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv. 44(2), 6:1–6:42 (2008)
14. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv. 44(2), 6:1–6:42 (Mar 2008)
15. Eskandari, M., Hashemi, S.: Metamorphic malware detection using control flow graph mining. International Journal of Computer Science and Network Security 11(12), 1–6 (2011)
16. Eskandari, M., Hashemi, S.: A graph mining approach for detecting unknown malwares. J. Vis. Lang. Comput. 23(3), 154–162 (2012)
17. Feily, M., Shahrestani, A., Ramadass, S.: A survey of botnet and botnet detection. In: Third International Conference on Emerging Security Information, Systems and Technologies. pp. 268–273 (2009)
18. Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D.: A survey of mobile malware in the wild. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. pp. 3–14 (2011)
19. Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R., Yan, X.: Synthesizing near-optimal malware specifications from suspicious behaviors. In: IEEE Symposium on Security and Privacy. pp. 45–60 (2010)
20. Garcia-Teodoro, P., Díaz-Verdejo, J.E., Maciá-Fernández, G., Vázquez, E.: Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security 28(1-2), 18–28 (2009)
21. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
22. Han, K.S., Kim, I.K., Im, E.: Malware classification methods using api sequence characteristics. In: Kim, K.J., Ahn, S.J. (eds.) Proceedings of the International Conference on IT Convergence and Security 2011, Lecture Notes in Electrical Engineering, vol. 120, pp. 613–626. Springer Netherlands (2012)
23. Islam, R., Tian, R., Batten, L.M., Versteeg, S.: Classification of malware based on integrated static and dynamic features. J. Network and Computer Applications 36(2), 646–656 (2013)
24. Jacob, G., Hund, R., Kruegel, C., Holz, T.: Jackstraws: Picking command and control connections from bot traffic. In: USENIX Security Symposium (2011)
25. Jeong, K., Lee, H.: Code graph for malware detection. In: ICOIN. pp. 1–5 (2008)
26. Khan, A., Yan, X., Wu, K.L.: Towards proximity pattern mining in large graphs. In: SIGMOD Conference. pp. 867–878 (2010)
27. Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic worm detection using structural information of executables. In: Valdes, A., Zamboni, D. (eds.) Recent Advances in Intrusion Detection, Lecture Notes in Computer Science, vol. 3858, pp. 207–226. Springer Berlin Heidelberg (2006)
28. Kruegel, C., Robertson, W., Valeur, F., Vigna, G.: Static disassembly of obfuscated binaries. In: USENIX Security Symposium. pp. 18–18 (2004)
29. Lee, J., Jeong, K., Lee, H.: Detecting metamorphic malwares using code graphs. In: SAC. pp. 1970–1977 (2010)
30. Li, Z., Liang, Y., Wu, Z., Tan, C.: Immunity based virus detection with process call arguments and user feedback. In: Bio-Inspired Models of Network,Information and Computing Systems. pp. 57–64 (2007)
31. Majumdar, A., Thomborson, C., Drape, S.: A survey of control-flow obfuscations. In: Bagchi, A., Atluri, V. (eds.) Information Systems Security, Lecture Notes in Computer Science, vol. 4332, pp. 353–356. Springer Berlin Heidelberg (2006)

32. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. 33(1), 31–88 (Mar 2001)
33. Perry, D.: Here comes the flood or end of the pattern file. Virus Bulletin (2008)
34. Preda, M.D., Christodorescu, M., Jha, S., Debray, S.: A semantics-based approach to malware detection. SIGPLAN Not. 42(1), 377–388 (Jan 2007)
35. Runwal, N., Low, R.M., Stamp, M.: Opcode graph similarity and metamorphic detection. Journal in Computer Virology 8(1-2), 37–52 (2012)
36. Shabtai, A., Moskovitch, R., Elovici, Y., Glezer, C.: Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. Inf. Secur. Tech. Rep. 14(1), 16–29 (Feb 2009)
37. Sherwood, T., Perelman, E., Hamerly, G., Calder, B.: Automatically characterizing large scale program behavior. SIGARCH Comput. Archit. News 30(5), 45–57 (Oct 2002)
38. Siddiqui, M., Wang, M.C., Lee, J.: A survey of data mining techniques for malware detection using file features. In: ACM Southeast Regional Conference. pp. 509–510 (2008)
39. Silva, S.S.C., Silva, R.M.P., Pinto, R.C.G., Salles, R.M.: Botnets: A survey. Comput. Netw. 57(2), 378–403 (Feb 2013)
40. Stumpf, S., Rajaram, V., Li, L., Wong, W.K., Burnett, M.M., Dietterich, T.G., Sullivan, E., Herlocker, J.L.: Interacting meaningfully with machine learning systems: Three experiments. Int. J. Hum.-Comput. Stud. 67(8), 639–662 (2009)
41. Tavallaee, M., Stakhanova, N., Ghorbani, A.A.: Toward credible evaluation of anomaly-based intrusion-detection methods. Trans. Sys. Man Cyber Part C 40(5), 516–524 (Sep 2010)
42. Wagener, G., State, R., Dulaunoy, A.: Malware behaviour analysis. Journal in Computer Virology 4(4), 279–287 (2008)
43. Wagner, D., Dean, D.: Intrusion detection via static analysis. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy. pp. 156– (2001)
44. Wang, X., Ding, X., Tung, A.K.H., Ying, S., Jin, H.: An efficient graph indexing method. In: ICDE. pp. 210–221 (2012)
45. Wong, W., Stamp, M.: Hunting for metamorphic engines. Journal in Computer Virology 2(3), 211–229 (2006)
46. Ye, Y., Wang, D., Li, T., Ye, D.: Imds: intelligent malware detection system. In: ACM SIGKDD. pp. 1043–1047 (2007)
47. You, I., Yim, K.: Malware obfuscation techniques: A brief survey. In: BWCCA. pp. 297–300. IEEE (2010)
48. Yu, Z., Tsai, J.J.: Intrusion detection: a machine learning approach, vol. 3. Imperial College Pr (2010)
49. Zhang, L., Yu, S., Wu, D., Watters, P.: A survey on latest botnet attack and defense. In: TrustCom. pp. 53–60 (2011)
50. Zhang, M.L., Zhou, Z.H.: Ml-knn: A lazy learning approach to multi-label learning. Pattern Recognition 40(7), 2038 – 2048 (2007)
51. Zhu, Y., Qin, L., Yu, J.X., Cheng, H.: Finding top-k similar graphs in graph databases. In: EDBT. pp. 456–467 (2012)